Learning to Understand Natural Language with Less Human Effort

Jayant Krishnamurthy

CMU-CS-15-110

May 2015

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Tom M. Mitchell, Chair
Eduard Hovy
Noah Smith
Luke Zettlemoyer

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the W. M. Keck Foundation under grant number DT123107 and the Air Force Research Laboratory under grant numbers FA87500810009, FA87501320003, and FA975009C0179 (sub #13705 with BBNT Solutions). The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **MAY 2015**	2. REPORT TYPE	3. DATES COVERED **00-00-2015 to 00-00-2015**
4. TITLE AND SUBTITLE **Learning to Understand Natural Language with Less Human Effort**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,School of Computer Science,Pittsburgh,PA,15213**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		

14. ABSTRACT

Learning to understand the meaning of natural language is an important problem within language processing that has the potential to revolutionize human interactions with computer systems. Informally, the problem specification is to map natural language text to a formal semantic representation connected to the real world. This problem has applications such as information extraction and understanding robot commands, and also may be helpful for other natural language processing tasks. Human annotation is a significant bottleneck in constructing language understanding systems. These systems have two components that are both constructed using human annotation: a semantic parser and a knowledge base. Semantic parsers are typically trained on individually-annotated sentences. Knowledge bases are typically manually constructed and given to the system. While these annotations can be provided in simple settings ??? specifically, when the knowledge base is small ??? the annotation burden quickly becomes unbearable as the size of the knowledge base increases. More annotated sentences are required to train the semantic parser and the knowledge base itself requires more annotations. Alternative methods to build language understanding systems that require less human annotation are necessary in order to learn to understand natural language in these more challenging settings. This thesis explores alternative supervision assumptions for building language understanding systems with the goal of reducing the annotation burden described above. I focus on two applications: information extraction and understanding language in physical environments. In the information extraction application, I present algorithms for training semantic parsers using only predicate instances from a knowledge base and an unlabeled text corpus. This algorithm eliminates the requirement for annotated sentences to train the semantic parser. I also present a new approach to semantic parsing that probabilistically learns a knowledge base from entity-linked text. This method reduces the amount of human annotation necessary to construct the knowledge base. Understanding language in physical environments breaks the assumptions of the approach above in that the learning agent must be able to perceive its environment to produce a knowledge base. I present a model that learns to map text to its real world referents that can be trained using annotated referents for entire texts, without requiring annotations of parse structure or the referents of individual words.

15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**187**	

Keywords: semantic parsing, natural language understanding, distant supervision, grounded language understanding

Abstract

Learning to understand the meaning of natural language is an important problem within language processing that has the potential to revolutionize human interactions with computer systems. Informally, the problem specification is to map natural language text to a formal semantic representation connected to the real world. This problem has applications such as information extraction and understanding robot commands, and also may be helpful for other natural language processing tasks.

Human annotation is a significant bottleneck in constructing language understanding systems. These systems have two components that are both constructed using human annotation: a semantic parser and a knowledge base. Semantic parsers are typically trained on individually-annotated sentences. Knowledge bases are typically manually constructed and given to the system. While these annotations can be provided in simple settings – specifically, when the knowledge base is small – the annotation burden quickly becomes unbearable as the size of the knowledge base increases. More annotated sentences are required to train the semantic parser and the knowledge base itself requires more annotations. Alternative methods to build language understanding systems that require less human annotation are necessary in order to learn to understand natural language in these more challenging settings.

This thesis explores alternative supervision assumptions for building language understanding systems with the goal of reducing the annotation burden described above. I focus on two applications: information extraction and understanding language in physical environments. In the information extraction application, I present algorithms for training semantic parsers using only predicate instances from a knowledge base and an unlabeled text corpus. This algorithm eliminates the requirement for annotated sentences to train the semantic parser. I also present a new approach to semantic parsing that probabilistically learns a knowledge base from entity-linked text. This method reduces the amount of human annotation necessary to construct the knowledge base. Understanding language in physical environments breaks the assumptions of the approach above in that the learning agent must be able to perceive its environment to produce a knowledge base. I present a model that learns to map text to its real world referents that can be trained using annotated referents for entire texts, without requiring annotations of parse structure or the referents of individual words.

Acknowledgments

This thesis would not have been possible without the support and guidance of my advisor, Tom Mitchell. His enthusiasm for research is contagious; one day I hope to instill the same enthusiasm in my own students.

I am also grateful to the members of the Read the Web group: Andy Carlson, Justin Betteridge, Matt Gardner, Derry Wijaya, Burr Settles, Partha Talukdar, Ndapa Nakashole, Alan Ritter, Anthony Platanios and Abulhair Saparov. Matt Gardner has been an especially close colleague and we have had many impromptu research discussions in his office. Bryan Kisiel also provided invaluable technical infrastructure without which much of this research would have been impossible.

The work on Logical Semantics with Perception is the product of a collaboration with Tom Kollar. I have to thank Tom for introducing me to the problem of grounded language understanding. This collaboration has been a tremendous influence on my thinking: I now realize that grounded language understanding is a key problem for the field of natural language semantics.

I would also like to thank my family. My parents, Anandi and Jagadisan Krishnamurthy, have always pushed me to work hard and excel, even though I usually proved quite resistant. My brother, Akshay Krishnamurthy, has also spent the last five years bursting in to my office to teach me machine learning theory, which is both educational and occasionally useful.

Contents

Chapter 1

Introduction

This thesis is about building computer programs that understand the meaning of natural language. This problem has captured the attention of artificial intelligence researchers since SHRDLU (Winograd, 1970) and remains an important problem with numerous practical applications such as personal voice assistants and information extraction systems. These applications will become increasingly important in the future because natural language is the easiest way to interact with robots and other embodied computer systems. Developing effective algorithms for understanding natural language is a critical problem with numerous real-world applications.

Understanding the meaning of language is also an important problem within the field of natural language processing. World knowledge often behaves as a constraint or cue for our syntactic interpretation. For example, consider the problem of identifying the correct prepositional phrase attachment in the following sentences:

1. I caught the butterfly with the net.
2. I caught the butterfly with the spots.

In sentence (1), the preposition "with" modifies "caught," while in sentence (2) it modifies "butterfly." Correctly disambiguating between these options requires understanding that, in the world, it is reasonable to find butterflies with spots, but not butterflies with nets. As another example, in coreference resolution, identifying that “Barack Obama” corefers with “president of the US” requires world knowledge about Obama’s political position. In both of these cases, language understanding is necessary to determine what world knowledge can be applied to resolve these ambiguities. These examples suggest the potential applications for natural language understanding (combined with world knowledge) in core NLP tasks.

The problem of understanding natural language can be factored into two subproblems. The first is *semantic parsing*, which is the problem of mapping natural language text to a formal representation of its meaning known as a *logical form*. For example, “president of the US” might be semantically parsed to $\lambda x.\textsc{presidentOf}(x, \textsc{US})$. A typical approach to this problem is to train a machine learning model using a corpus of text with annotated logical forms. The second is *modelling the world*, which is the problem of constructing a formal representation of the real world. This representation describes the objects that exist in the world, along with their properties and relations between them. Throughout this thesis, I assume that this component is provided by a logical knowledge base containing entities (e.g., $\textsc{Obama}$) and predicate instances (e.g., $\textsc{presidentOf}(\textsc{Obama}, \textsc{US})$). This knowledge base is typically manually constructed and provided to the system, though it can also be learned automatically. Logical forms can be interpreted as queries against the knowledge base, and evaluating a logical form against the knowledge base produces its *denotation*, which is typically a set of entities or a truth value. In the running example, the denotation of $\lambda x.\textsc{presidentOf}(x, \textsc{US})$ is the set of all US presidents, including $\textsc{Obama}$. This approach to understanding natural language has been successfully applied to several question answering tasks (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005, 2007).

A noteworthy aspect of this problem definition is that *language understanding is defined relative to a knowledge base*. As in model-theoretic semantics (Dowty et al., 1981), understanding is defined as the problem of mapping from text to elements of the knowledge base. Therefore, the choice of the knowledge base determines the subset of language that can be understood. For example, above, if the knowledge base did not have a PRESIDENTOF predicate, it would not be possible to represent the meaning of "president of the US." Any inference capabilities of the system are also dependent on the knowledge base. In this formulation, the choice of knowledge base has a tremendous impact on the capabilities of a language understanding system.

The object of this thesis is to enable us to understand natural language in more challenging settings by eliminating the obstacle of *human annotation*. Current approaches to language understanding require humans to provide (1) per-sentence semantic annotations to train the semantic parser and (2) a knowledge base. Per-sentence semantic annotations are labor intensive to create and the number of annotations necessary grows with the number of predicates in the knowledge base. Unfortunately, a broad knowledge base is necessary to understand language about a broad variety of topics. Training semantic parsers for large knowledge bases requires us to move beyond using manually labeled data. (Indeed, the use of additional data in recent work on semantic parsing to Freebase suggests that current data sets are too small (Berant and Liang, 2014; Cai and Yates, 2013b).) Knowledge bases are similarly challenging to build and even the largest knowledge bases remain incomplete (Bollacker et al., 2008; Lenat, 1995). In other settings, such as understanding language in the context of a particular physical environment, the knowledge base depends on the environment and therefore must be automatically produced by the language understanding agent in order to generalize to new environments. These cases suggest that moving beyond a manually constructed knowledge base is also necessary. Eliminating the human annotation bottleneck is required to enable us to understand language in these more challenging settings.

In this thesis, I propose algorithms for language understanding that eliminate the need for human annotation of both the knowledge base and the language's semantics. The thesis of this research is that *language understanding systems can be constructed for both grounded and ungrounded settings using existing resources with little to no additional human annotation*. I support this thesis by providing a variety of learning algorithms that train language understanding systems using easily-obtainable forms of supervision, such as unlabeled text, instead of human annotated data. I first consider the ungrounded setting, in which the language understanding agent is not embodied in a physical environment. Chapter 4 describes a training algorithm for this setting where a knowledge base is given, but no per-sentence semantic annotations are necessary. Chapter 5 further reduces the dependence on a knowledge base by instead learning predicate instances automatically from unannotated text. These chapters demonstrate that, given a knowledge base, a language understanding system can be constructed with no additional human annotation. Chapter 6 considers the grounded case, where the language being interpreted refers to objects in a physical environment perceived by the agent. The algorithm in this chapter replaces both forms of annotation described above with annotated denotations, which intuitively are the real-world referents of a statement. This chapter demonstrates that simple annotations, in the form of denotations, are sufficient to train grounded language understanding systems.

1.1 Axes of Variation

Each chapter of this work considers a distinct approach to understanding natural language that can be characterized by a small number of axes. I highlight these axes here; later chapters will further elaborate on the benefits and limitations of each position. Table 1.1 summarizes the positioning of prior work and the chapters of this thesis along these axes.

1. **Learning the knowledge base** – Language understanding is relative to a knowledge base, yet constructing broad coverage knowledge bases remains an open problem. Thus, a natural question is whether the knowledge base can be learned automatically. This work considers two distinct settings. The first is to simply assume that a large knowledge base is given to the program. This approach is typical in prior work (e.g., (Zettlemoyer and Collins, 2005)) and is taken in Chapter 4. The second is to learn the knowledge base while learning to understand language. This approach is taken in Chapters 5 and 6. Somewhat unintuitively, learning the knowledge base actually *simplifies* the language understanding

problem in an important respect: the program no longer has to learn a correspondence between words and knowledge base predicates.

2. **Predicate vocabulary** – Manually created knowledge bases have a closed (and relatively limited) vocabulary of formal predicates. For example, NELL (Carlson et al., 2010) contains ~ 600 predicates, such as $\textsc{athlete}(x)$ and $\textsc{teamPlaysSport}(x, y)$. The set of predicates restricts the language that is understandable using the knowledge base. An open predicate vocabulary can circumvent this limitation by treating every word as its own formal predicate (Banko et al., 2007; Riedel et al., 2013). Chapters 5 and 6 both use open predicate vocabularies.

3. **Semantic parser supervision** – A key issue addressed in this work is identifying natural and useful sources of supervision for training semantic parsers. I consider two distinct types of supervision: unlabeled text and denotations. Unlabeled text is used in Chapters 4 and 5, which both consider the ungrounded setting. Unlabeled text is easy to obtain, and can be used as training data if we assume that it expresses true statements about the world. Chapter 6 considers the grounded setting and uses annotated denotations, which are the real-world referents for natural language statements. Although denotations are more challenging to obtain than unlabeled text, they remain easier than full knowledge bases and annotated logical forms. The grounded setting also seems to require at least a few annotated denotations in order to learn the mapping from words to real-world objects.

4. **Syntactic information** – Current work on semantic parsing typically assumes that no syntactic information is provided to the learning agent. I also make this assumption in Chapter 6. Although this assumption is reasonable when the language is simple, syntactic information can provide an advantage when language becomes more complex. Chapter 4 finds that syntactic information is extremely helpful for semantically analyzing complex language. Chapter 5 takes this view to the extreme: it produces semantic parses using a rule-based transformation of syntactic parses.

These four axes are intended to describe the amount and type of human annotation necessary to train the systems in each chapter. The primary axes are 1 and 3, which directly characterize the two main forms of human annotation provided to language understanding systems. The other two axes interact with these. Specifically, if no knowledge base is given (axis 1), an open predicate vocabulary seems necessary (axis 2). Syntactic information (axis 4) turns out to be very useful when less informative annotations are given to train the semantic parser (axis 3).

1.2 Chapter Overview

This thesis is organized as follows. Two introductory chapters provide context for the contributions of this thesis. Chapter 2 provides an introduction to semantic parsing, a key subproblem of language understanding. This chapter also defines and standardizes the terminology used throughout this thesis. Chapter 3 describes relevant prior work.

Chapter 4 introduces a distantly-supervised algorithm for training a semantic parser. This algorithm uses a knowledge base and a large corpus of unlabeled sentences to train a semantic parser, without any semantically-annotated sentences. The second part of this chapter extends this distant supervision approach to incorporate syntactic information, resulting in a full syntactic parser for English that also produces semantic analyses for portions of the input sentence. Both parsers from this chapter are effective information extraction tools and can be used to answer natural language queries against the NELL knowledge base.

Chapter 5 introduces a new approach to semantic parsing based on learning a knowledge base instead of mapping text to an existing knowledge base. This approach uses manually-defined rules to convert a sentence's syntactic parse into a logical form containing predicates derived from the words in the sentence. The denotations of these predicates are learned automatically from an entity-linked text corpus using probabilistic matrix factorization. The combined system is capable of producing the same model-theoretic semantics as a traditional semantic parser (as in Chapter 4), but is not restricted to using predicates from a particular knowledge base. On

	Learning the knowledge base	Predicate vocabulary	Semantic parser supervision	Syntactic information
Typical in prior work (c.f., (Berant and Liang, 2014; Berant et al., 2013; Cai and Yates, 2013b; Kwiatkowski et al., 2011, 2013; Liang et al., 2011; Zettlemoyer and Collins, 2005, 2007))	Knowledge base is given	Closed	Annotated logical forms or denotations	Semantic parser trained without any syntactic information
Chapter 4: Training semantic parsers with distant supervision	Knowledge base is given	Closed	Unlabeled text	Dependency parses and CCG treebank used to supervise the training of a semantic parser
Chapter 5: Semantic parsing with an open predicate vocabulary	Knowledge base is automatically learned	Open	Entity-linked text	CCG syntactic parser combined with manually-specified rules to produce semantic parses; no learned semantic parser.
Chapter 6: Semantic parsing in grounded environments	Learned classifiers produce a knowledge base from an environment	Open	Denotations	Semantic parser trained without any syntactic information

Table 1.1: Positioning of the thesis chapters and prior work with respect to the main axes of variation of this thesis.

a question answering task, this approach is capable of answering many questions that queries against Freebase cannot.

Chapter 6 describes Logical Semantics with Perception (LSP), a model for understanding natural language in grounded environments. For example, given the statement “blue mug on the table,” LSP can identify the segment of an image depicting a blue mug on a table. This model combines a semantic parser with a perception function that produces a logical knowledge base from a physical environment. This chapter also describes a weakly-supervised algorithm for training both the semantic parser and perception function from annotated denotations. A question answering evaluation demonstrates that this weakly-supervised has comparable performance to a fully-supervised baseline while requiring significantly less manual annotation.

Finally, Chapter 7 summarizes the lessons from this work and suggests directions for future work.

Chapter 2

Semantic Parsing with Combinatory Categorial Grammar

Semantic parsing is the problem of mapping natural language text to a formal representation of its meaning, known as a *logical form*. This problem is one of the two key subproblems of natural language understanding: it is the problem of mapping from natural language text to an agent's internal representation of the world, given by a logical knowledge base. This chapter presents a simple algorithm for learning a semantic parser that forms the basis for learning parsers in later chapters of this dissertation. These later chapters weaken the supervision assumptions used in this chapter and derive new learning algorithms for these settings. The algorithm presented in this chapter is a highly-simplified variant of published algorithms (Zettlemoyer and Collins, 2005, 2007).

This chapter considers the problem of learning a semantic parser given (1) a logical knowledge base and (2) an annotated training data set of language/logical form pairs. This problem setting has been successfully used in many language understanding applications, including answering natural language questions against a database (Zelle and Mooney, 1996), coaching RoboCup soccer (Ge and Mooney, 2009; Wong and Mooney, 2007), giving robots navigational instructions (Artzi and Zettlemoyer, 2013; Chen and Mooney, 2011; Matuszek et al., 2010), and parsing temporal expressions (Lee et al., 2014). For concreteness, this chapter focuses on an imaginary natural language database query application, with input/output of the following form:

- Input (language): “what cities are in Texas?”
- Output (logical form): $\lambda x.\textsc{city}(x) \wedge \textsc{locatedIn}(x, \textsc{Texas})$.

The semantic parser built in this chapter has two components: a heuristically-defined grammar, and a machine-learned parsing model. The grammar defines a set of possible logical forms for every natural language sentence. It will be defined using Combinatory Categorial Grammar (CCG), a grammar formalism that tightly couples syntax and semantics, naturally facilitating semantic parsing. However, the grammar will generally permit multiple logical forms to be produced for every natural language sentence. The role of the parsing model is to select the correct logical form from amongst these possibilities. The presented approach will estimate both the grammar and the parsing model from a data set of sentences with annotated logical forms, as in the example above.

The structure of this chapter is as follows. Section 2.1 describes the CCG grammar formalism. Section 2.2 describes a simple method for using CCG to build a semantic parser. (Improvements on this simple method are described later in Chapter 3.) Finally, Section 2.3 discusses the benefits and limitations of the presented approach to semantic parsing.

2.1 Combinatory Categorial Grammar

Combinatory Categorial Grammar (Steedman, 2000; Steedman and Baldridge, 2011) is a grammar formalism that has been used extensively for both wide-coverage syntactic parsing (Auli and Lopez, 2011a; Clark and Curran, 2007a; Hockenmaier, 2003a; Hockenmaier and Steedman, 2002b; Lewis and Steedman, 2014) and semantic parsing (Artzi and Zettlemoyer, 2013; Artzi et al., 2014; Cai and Yates, 2013a,b; Krishnamurthy and Kollar, 2013; Krishnamurthy and Mitchell, 2012, 2014; Kwiatkowski et al., 2010, 2011, 2013; Lee et al., 2014; Matuszek et al.,

Syntactic category	Part of speech	Example usage
N/N	Adjective	*good* person : N
NP/N	Determiner	*the* person : NP
$S\backslash N$	Intransitive Verb	I *ate* : S
$(S\backslash N)/N$	Transitive Verb	I *ate* strawberries : S
$(S\backslash N)/(S\backslash N)$	Adverb	I *quickly* ate : S
$(N\backslash N)/N$	Preposition	person *in* Texas : N
$(S\backslash N)\backslash(S\backslash N)/N$	Preposition	ate *in* Texas : $S\backslash N$

Table 2.1: Common syntactic categories in CCG with explanations of their usage.

2012; Reddy et al., 2014b; Zettlemoyer and Collins, 2005, 2007). CCG's use for semantic parsing is motivated by its tight coupling of syntax and semantics: words and phrases in CCG are mapped to both a syntactic and a semantic representation, and parsing naturally combines both of these representations to produce representations of larger phrases. Several other considerations also motivate the use of CCG for semantic parsing. First, it is well-studied by linguists, and consequently many linguistic phenomena (e.g., various forms of coordination) have established analyses (Steedman and Baldridge, 2011). Second, CCG has a wide-coverage syntactic treebank (Hockenmaier and Steedman, 2002a) that can be used to assist with semantic parsing (as in Chapters 4 and 5). The wide-coverage syntactic CCG parsers built with this treebank are another useful resource (Clark and Curran, 2007a).

Structurally, CCG is a lexicalized grammar formalism, meaning it has a large set of syntactic categories that are combined using a small number of parsing operations. Hence, much of the grammatical information is encoded in the syntactic categories themselves. The intuition of CCG is that, syntactically and semantically, words and phrases behave *like functions*. For example, an adjective can combine with a noun to produce another noun, as in the phrase "small town." Thus, the set of adjectives can be naturally identified with the set of functions that take a noun argument and return a noun. Syntactic categories in CCG explicitly encode such argument type specifications. The meaning of a particular adjective is then a particular function in this set – that is, one function represents the meaning of "small," another "large," etc. In CCG, this function is the word's logical form. Parsing in CCG combines adjacent words and phrases using a small number of operations, such as functional application and composition. Hence, to parse the phrase "small town," the "small" function is applied to the argument "town." This operation naturally derives both a syntactic category and logical form for the combined phrase.

2.1.1 Syntactic Categories

CCG has two kinds of syntactic categories: basic and functional. Basic categories represent phrases that do not accept arguments; typically, this set contains only N for noun, NP for noun phrase, PP for prepositional phrase, and S for sentence.[1] Functional categories represent phrases that accept arguments. These categories are written as $X\backslash Y$ or X/Y where both X and Y are syntactic categories. Both of these categories indicate functions that accept an argument of category Y and return a value of category X. The direction of the slash determines where the argument must appear: $/$ indicates an argument on the right, and $\backslash$ indicates an argument on the left. For example, the category $S\backslash N$ represents an intransitive verb; it is a function that accepts a noun on the left and returns a sentence. Functional categories can also be nested to represent more complex phenomena. For example, the category $(S\backslash N)/N$ represents a transitive verb; it is a function that first accepts a noun on the right, then a noun on the left, and returns a sentence. Other common syntactic categories are listed in Table 2.1.

2.1.2 Semantics

The semantics of a sentence are given by its *logical form*, which is a statement in a typed lambda calculus. In this thesis, logical forms can be interpreted as queries against a knowledge base. They are defined in a lambda calculus with two basic semantic types, e for entity and t for truth value. Functional types are built from these basic types, such as $\langle e, t\rangle$, representing functions from entities to truth values.

[1] This set can be chosen to support specific applications. For example, wide-coverage syntactic parsers assign separate basic syntactic categories to punctuation symbols.

Logical forms are composed of the following items:

- **Entities** such as $\textsc{Texas}$, of type e. The set of entities is defined by the knowledge base.
- **Predicates** are functions from entities to truth values, such as $\textsc{city}(x)$ and $\textsc{locatedIn}(x, y)$. One argument predicates are called *categories* and have type $\langle e, t\rangle$. Two argument predicates are called *relations* and have type $\langle e, \langle e, t\rangle\rangle$; these will be written as two-argument functions for clarity, e.g., $\textsc{r}(x, y)$, not $\textsc{r}(x)(y)$. The set of predicates is also defined by the knowledge base.
- **Variables** of any type. Throughout this thesis, the variables w, x, y, z denote entities and f, g, h denote functions.
- **Logical connectives** such as conjunction $\wedge$, disjunction $\vee$, and implication $\rightarrow$, all of type $\langle t, \langle t, t\rangle\rangle$. Logical forms may also include negation $\neg$, with type $\langle t, t\rangle$ and equality $=$, with type $\langle e, \langle e, t\rangle\rangle$.
- **Quantification**, including both existential $\exists$ and universal $\forall$ quantifiers. Quantifiers have type $\langle\langle e, t\rangle, t\rangle$.
- **Lambda expressions** representing functions. For example, $\lambda x.\neg\textsc{city}(x)$ denotes the function that takes an entity argument x and returns true if x is not a $\textsc{city}$. Lambdas can be used to create functions with many different type specifications. For example, the expression $\lambda f.\lambda x.\textsc{city}(x) \wedge f(x)$ has type $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$.

Logical forms are programs that can be evaluated on a knowledge base to return a *denotation*. If the logical form has type t, the denotation is a truth value; if it has type $\langle e, t\rangle$, the denotation is a set of entities. The precise value of the denotation depends on the predicate instances in the knowledge base. For example, consider evaluating $\lambda x.\textsc{city}(x) \wedge \textsc{locatedIn}(x, \textsc{Texas})$ on a knowledge base that contains $\textsc{city}(\textsc{Plano})$ and $\textsc{locatedIn}(\textsc{Plano}, \textsc{Texas})$. The denotation of this logical form is the set of entities x for which it returns true, which in this case is the set $\{\textsc{Plano}\}$. In question answering applications, the denotation is the answer to the posed question.

The language for logical forms used in this thesis is a fairly typical language for representing database queries. Other work commonly includes real numbers and functions that operate on real numbers, such as COUNT or LARGEST (e.g, (Zettlemoyer and Collins, 2005)). These items are omitted in this thesis because the knowledge bases used do not contain numerical information. Naturally, other applications of semantic parsing may use different types and additional items; for example, interpreting robot commands may require an action type and actions such as TURNLEFT.

A correspondence between syntactic categories and semantic types ensures that parsing produces well-typed logical forms. A standard choice is to assign nouns N, noun phrases NP and prepositional phrases PP the type $\langle e, t\rangle$, and sentences S the type t. Note that a function f of type $\langle e, t\rangle$ is isomorphic to a set of entities, specifically, the set of entities for which f returns true. Thus, the denotation of a noun is the set of entities it refers to and the denotation of a sentence is either true or false.[2] This association naturally extends to functional syntactic categories. For example, adjectives N/N have semantic type $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$.

2.1.3 Lexicon

A CCG grammar is defined by a *lexicon*, which is a mapping from words to syntactic categories and logical forms. The lexicon, along with the parsing operations described below, defines the set of possible parses for every natural language statement. An example lexicon may include the following entries:

[2] This correspondence results in a simple model-theoretic semantics for language (Dowty et al., 1981). Although this simple semantics is sufficient to represent the phenomena considered in this thesis, many phenomena – such as modelling the beliefs of other agents – require a more complex correspondence between syntactic and semantic types. However, many knowledge bases available today do not represent these phenomena, so there is little immediate value in a more complex semantic analysis. This semantics is also a reasonable starting point that can be extended in future work.

town	:=	N	:	$\lambda x.\textsc{city}(x)$
Texas	:=	N	:	$\lambda x.x = \textsc{Texas}$
Alex	:=	N	:	$\lambda x.x = \textsc{Alex}$
large	:=	N/N	:	$\lambda f.\lambda x.f(x) \wedge \textsc{major}(x)$
in	:=	$(N\backslash N)/N$	:	$\lambda f.\lambda g.\lambda x.\exists y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x, y)$
in	:=	(PP/N)	:	$\lambda f.f$
lives	:=	$(S\backslash N)/PP$	:	$\lambda f.\lambda g.\exists x, y.f(y) \wedge g(x) \wedge \textsc{HasResidence}(x, y)$
is	:=	$(S\backslash N)/N$	:	$\lambda f.\lambda g.\exists x.f(x) \wedge g(x)$

During parsing, these lexicon entries are used to initialize the set of possible syntactic categories and logical forms for each word. Note that a single word may have multiple lexicon entries with distinct syntactic categories and logical forms; parsing must disambiguate between these options. Further note that the lexicon entries with functional syntactic categories have a one-to-one mapping between arguments of the syntactic category and arguments of the logical form. For example, in the lexicon entry for "lives," the argument f corresponds to the PP and the argument g corresponds to the N. During parsing, f and g will be assigned values that represent the semantics of the corresponding syntactic arguments.

2.1.4 Parsing

Parsing in CCG combines adjacent categories using a small number of parsing operations, or *combinators*. These combinators simultaneously operate on syntactic categories and logical forms, jointly deriving the syntax and semantics of the parsed text. A parse in CCG is known as a *derivation*.

Application

The most commonly-used combinator in CCG is functional application:

$$\begin{array}{lll} X/Y : f & Y : g & \Longrightarrow\ X : f(g) \\ Y : g & X\backslash Y : f & \Longrightarrow\ X : f(g) \end{array}$$

The first rule, forward application, states that the category X/Y can be applied to the category Y on the right, returning category X. Similarly, the logical form f is applied to g to produce the logical form for the returned category. Given the sample lexicon above, this rule can be used to produce the following parse:

$$\dfrac{\dfrac{\text{major}}{N/N : \lambda f.\lambda x.f(x) \wedge \text{LARGE}(x)} \quad \dfrac{\text{town}}{N : \lambda x.\text{CITY}(x)}}{N : \lambda x.\text{CITY}(x) \wedge \text{LARGE}(x)}$$

The second rule, backward application, is analogous to the first, except that the argument appears on the left. A grammar that only uses functional application is known as a Categorial Grammar (Ajdukiewicz, 1935; Bar-Hillel, 1953).

Composition

CCG extends Categorial Grammar with a number of additional combinators, such as forward and backward composition:

$$\begin{array}{lll} X/Y : f & Y/Z : g & \Longrightarrow\ X/Z : \lambda z.f(g(z)) \\ Y\backslash Z : g & X\backslash Y : f & \Longrightarrow\ X\backslash Z : \lambda z.f(g(z)) \end{array}$$

The first composition rule states that the category X/Y can compose with the category Y/Z to produce the category X/Z. This rule makes intuitive sense: Y/Z is a function that takes a Z and produces a Y, so applying X/Y to the resulting Y gives us a function from Z to X, or the category X/Z. The second composition rule is analogous to the first but in the opposite direction. A simple example of composition is:

$$\dfrac{\dfrac{\text{really}}{(S\backslash N)/(S\backslash N) : \lambda f.f} \quad \dfrac{\text{lives}}{(S\backslash N)/PP : \lambda f.\lambda g.\exists x, y.f(y) \wedge g(x) \wedge \text{HASRESIDENCE}(x, y)}}{(S\backslash N)/PP : \lambda f.\lambda g.\exists x, y.f(y) \wedge g(x) \wedge \text{HASRESIDENCE}(x, y)}$$

In many cases, composition is used with coordination or type-raising to parse sentences that cannot be parsed with only functional application. Examples of these uses are given below.

Composition also has several variants that compose deeper arguments. For example, the following rule composes a one-argument category with a two-argument category:

$$X/Y : f \quad (Y/W)/Z : \lambda z.\lambda w.g(z)(w) \quad \Longrightarrow (X/W)/Z : \lambda z.\lambda w.f(g(z)(w))$$

Another variant, crossed composition, can be used to compose syntactic categories with different argument directionalities:

$$\begin{aligned} X/Y : f \quad & Y\backslash Z : g \implies X\backslash Z : \lambda z.f(g(z)) \\ Y/Z : g \quad & X\backslash Y : f \implies X/Z : \lambda z.f(g(z)) \end{aligned}$$

Type-Raising

Type-raising is another permitted combinator. This is a unary combinator that can be applied to a category in isolation:

$$X : f \implies Y/(Y\backslash X) : \lambda g.g(f)$$

Type-raising is often used with composition to combine categories with their arguments in a different order than the one imposed by the category. For example, these two operations can be used to combine a verb with its subject before its object:

$$\begin{array}{cc}
\text{Alex} & \text{lives} \\ \hline
N : \lambda x.x = \textsc{Alex} & (S\backslash N)/PP : \\ \cline{1-1}
S/(S\backslash N) : \lambda f.f(\lambda x.x = \textsc{Alex}) & \lambda f.\lambda g.\exists x, y.f(y) \land g(x) \land \textsc{HasResidence}(x, y) \\ \hline
\multicolumn{2}{c}{S/PP : \lambda h.(\lambda f.\lambda g.\exists x, y.f(y) \land g(x) \land \textsc{HasResidence}(x, y))(h)(\lambda x.x = \textsc{Alex})} \\
\multicolumn{2}{c}{\text{simplified:} \quad S/PP : \lambda h.\exists x, y.h(y) \land x = \textsc{Alex} \land \textsc{HasResidence}(x, y)}
\end{array}$$

This parse first type-raises “Alex,” then composes the result with “lives” to obtain the desired result. The first version of the resulting category shows the lambda calculus expression created by directly following the rules for type-raising and composition. The second version simplifies this expression according to the rules of lambda calculus.

Coordination

Coordination is represented in CCG by syntactic categories of the form $(X\backslash X)/X$, such as $(N\backslash N)/N$ to coordinate nouns, or $((S\backslash N)\backslash(S\backslash N))/(S\backslash N)$ to coordinate verbs. For example:

$$
\begin{array}{ccc}
\underline{\text{Alex}} & \underline{\text{and}} & \underline{\text{Texas}} \\
N : \lambda x.x = \textsc{Alex} & (N\backslash N)/N : \lambda f.\lambda g.\lambda x.f(x) \vee g(x) & N : \lambda x.x = \textsc{Texas} \\
\cline{2-3}
 & \multicolumn{2}{c}{N\backslash N : \lambda g.\lambda x.f(x) \vee x = \textsc{Texas}} \\
\hline
\multicolumn{3}{c}{N : \lambda x.x = \textsc{Alex} \vee x = \textsc{Texas}}
\end{array}
$$

Additional Combinators

Finally, it is often useful to include additional unary and binary combinators in a CCG semantic parser. These combinators are typically specially designed to simplify parsing for a particular application. For example, wide-coverage syntactic parsers must process punctuation marks. The following two rules can be used to combine commas with nouns:

$$
\begin{array}{lll}
N : f \quad ,: g & \Longrightarrow & N : f \\
\quad\quad\quad ,: h & \Longrightarrow & (N\backslash N)/N : \lambda f.\lambda g.\lambda x.f(x) \vee g(x)
\end{array}
$$

The first binary combinator allows a noun to absorb a comma on the right with no effect on its semantics. In many cases, this is an appropriate way to semantically parse a comma. However, a comma may also indicate a coordination, as in a comma-separated list. The second unary combinator allows a comma to be type-changed to a coordination between nouns. Another example of additional combinators are the binary rules used by the parser in Section 4.2 to analyze noun compounds such as “Texas city.”

2.1.5 Parsing Algorithms

Combinatory Categorial Grammar belongs to the class of so-called mildly context-sensitive grammar formalisms, meaning it is capable of generating non-context-free languages (Weir and

Lexicon:	**Derivations:**
foo := $A/A/A$	foo foo : $A/A/A/A$
foo := $A/B/A$	foo foo : $A/B/A/A$
	foo foo : $A/A/B/A$
	foo foo : $A/B/B/A$

Figure 2.1: A CCG grammar that may require exponential time to parse with the CKY algorithm. The string containing "foo" repeated n times has 2^n possible syntactic categories for the parse tree root, of the form $A/[A$ or $B]/.../A$.

Joshi, 1988). Due to its enhanced generative power, parsing CCG with the CKY algorithm (Manning and Schütze, 1999) may require time exponential in the length of the sentence. Figure 2.1 illustrates a simple grammar for which this is the case. The fundamental problem is that composition can be used to grow the number of arguments to a category during parsing, resulting in an exponential number of possible syntactic categories at the root of the parse tree. However, alternative algorithms exist for parsing CCG in $O(n^6)$ time (Kuhlmann and Satta, 2014; Vijay-Shanker and Weir, 1990, 1993); these algorithms do not explicitly represent all possible syntactic categories at each chart entry, allowing them to avoid an exponential running time.[3]

Despite these theoretical limitations, in practice, CCG parsing is performed using the CKY algorithm. There are several common optimizations used to make parsing with CKY more tractable:

1. **Restrict the Grammar.** Define a subset of the full grammar that contains a finite set of syntactic categories, such as the categories that appear in a treebank. Restricting CCG to a finite set of syntactic categories results in a context-free grammar, guaranteeing that the CKY algorithm runs in polynomial time (Fowler and Penn, 2010). This restriction is relatively minor, as the complex categories produced by repeated compositions are typically not meaningful or useful.

2. **Normal Form.** Due to its additional combinators, CCG often permits multiple syntactic

[3] Note that categorial grammar – CCG restricted to only functional application – is context-free and therefore can be parsed in $O(n^3)$ time using CKY.

parses for a sentence that result in equivalent logical forms. For example, there are two semantically equivalent parses of “clean major town:”

$$\begin{array}{ccc}
\text{clean} & \text{major} & \text{town} \\
\hline
N/N : \lambda f.\lambda x.f(x) \wedge \textsc{clean}(x) & N/N : \lambda f.\lambda x.f(x) \wedge \textsc{major}(x) & N : \lambda x.\textsc{city}(x) \\
\cline{2-3}
 & \multicolumn{2}{c}{N : \lambda x.\textsc{city}(x) \wedge \textsc{major}(x)} \\
\hline
\multicolumn{3}{c}{N : \lambda x.\textsc{city}(x) \wedge \textsc{major}(x) \wedge \textsc{clean}(x)}
\end{array}$$

$$\begin{array}{ccc}
\text{clean} & \text{major} & \text{town} \\
\hline
N/N : \lambda f.\lambda x.f(x) \wedge \textsc{clean}(x) & N/N : \lambda f.\lambda x.f(x) \wedge \textsc{major}(x) & N : \lambda x.\textsc{city}(x) \\
\cline{1-2}
\multicolumn{2}{c}{N/N : \lambda x.\textsc{clean}(x) \wedge \textsc{major}(x)} & \\
\hline
\multicolumn{3}{c}{N : \lambda x.\textsc{city}(x) \wedge \textsc{major}(x) \wedge \textsc{clean}(x)}
\end{array}$$

This example illustrates the problem of *spurious ambiguity*, which occurs when a phrase that can be parsed using only functional application also permits additional, semantically-equivalent parses using other combinators. These additional parses reduce parser efficiency (Clark and Curran, 2007a).

A CCG parser can be constrained to produce only *normal form* syntactic derivations to eliminate spurious ambiguity (Eisner, 1996). Informally, a syntactic derivation is normal form if it only uses composition and type-raising where necessary. Normal form derivations can be obtained by restricting the combination of categories produced by composition. Specifically, a category produced by composition cannot be used in a subsequent functional application as the primary functor, i.e., the function being applied. In the second example above, this rule would prevent the composed N/N category for “clean major” from combining with “town.” This rule is sufficient to completely eliminate spurious ambiguity in grammars containing only functional application and composition, but may only partially eliminate spurious ambiguity if other combinators are allowed.

3. **Supertagging.** Even with the above restrictions, wide-coverage CCG grammars permit a large number of parses for any given sentence. These grammars have hundreds of syntactic

categories and their lexicons contain numerous entries for certain words. For example, the CCGbank lexicon contains 79 entries for “in.” Consequently, the space of parses for a sentence can be too large to search in practice.

Supertagging is a method for improving parser efficiency by pruning the possible syntactic categories for each token in a sentence before parsing (Bangalore and Joshi, 1999; Clark, 2002; Clark and Curran, 2004). Supertagging is performed by training a model to predict a set of syntactic categories for each token using features of the surrounding tokens. Subsequent parsing is restricted to the predicted set of syntactic categories. This technique is effective because, even though a word may have many possible syntactic categories, some categories are much more likely to appear in a particular sentence than others.

In this thesis, every semantic parser employs the context-free grammar optimization and the wide-coverage parser described in Chapter 4 employs all of the above optimizations.

2.2 Building a Semantic Parser

This section describes how to build a semantic parser given a data set of sentences paired with logical forms. The approach described is somewhat simplistic, but represents a reasonable starting point for more advanced work on semantic parsing. Improvements on this basic approach are described in Section 3.1.

The input to this approach is a knowledge base and a collection of sentences paired with logical forms. The approach has two steps. First, it defines a grammar for the semantic parser by creating a CCG lexicon and selecting combinators. This step is performed by a collection of manually-defined heuristics and results in a grammar that permits many CCG parses for any given sentence. The second step trains a machine learning model to select the correct parse from among the many possibilities. This step formulates semantic parsing as a structured prediction problem and trains a model using the structured perceptron algorithm.

2.2.1 Data

This approach to building a semantic parser has two inputs: a knowledge base and a collection of sentences with annotated logical forms. The knowledge base $K = (E, R, C, \Delta)$, contains entities $e \in E$, category predicates $c \in C$, relation predicates $r \in R$ and predicate instances $\{c(e), r(e_1, e_2)\} \in \Delta$. The knowledge base defines the vocabulary of predicates and entities for logical forms, and contains the predicate instances required to evaluate logical forms. (Evaluating logical forms will not be necessary for training, but is typically desirable for downstream applications.) Although not explicitly listed here, the knowledge base may also contain other useful information, such as a type hierarchy or known names of entities.

The data for training is a collection of n tuples, $\{(s_i, \ell_i)\}_{i=1}^{n}$, of sentences s_i paired with logical forms ℓ_i. Each sentence is a sequence of words (not necessarily a grammatically correct sentence), and each logical form is a lambda calculus statement. For concreteness, this section focuses on the problem of querying a database with natural language. In this setting, the sentence is a natural language query, e.g., $s_i =$ "cities in Texas," and the logical form is a database query, e.g., $\ell_i = \lambda x.\textsc{city}(x) \wedge \textsc{locatedIn}(x, \textsc{Texas})$. These annotated sentences are used to both derive the grammar of the parser and train the structured prediction model.

2.2.2 Grammar Induction

The grammar $G = (\Lambda, C)$ of a semantic parser has two components, a lexicon Λ and a set of combinators C. These components together define the set of possible parses for each sentence. The challenging portion of grammar induction is inducing the lexicon, as the combinators can simply be selected from the set of possibilities described in Section 2.1.4.

Lexicon induction produces a CCG lexicon Λ for the semantic parser given a data set of sentences paired with logical forms. A simple approach to lexicon induction is to define a collection of heuristic rules that instantiate lexicon entries from training examples. For example, to correctly parse the "cities in Texas" example above, the CCG lexicon should ideally contain the following entries:

$$\begin{array}{llll}
\text{cities} & := & N & : & \lambda x.\textsc{city}(x) \\
\text{Texas} & := & N & : & \lambda x.x = \textsc{Texas} \\
\text{in} & := & (N\backslash N)/N & : & \lambda f.\lambda g.\lambda x.\exists y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x, y)
\end{array}$$

These entries can be generated by part-of-speech tagging the text then applying the following heuristics:

1. Generate a lexicon entry with syntactic category N mapping every noun in the input to every category c and entity e in the annotated logical form. The semantics of these entries have the form $\lambda x.c(x)$ or $\lambda x.x = e$.
2. Generate a lexicon entry with syntactic category $(N\backslash N)/N$ mapping every preposition in the input to every relation in the annotated logical form. The semantics of this entry has the form $\lambda f.\lambda g.\lambda x.\exists y.f(y) \wedge g(x) \wedge r(x, y)$.

These heuristics are simple examples of the kinds of rules that can be employed. Real applications typically employ 10-15 such rules (Krishnamurthy and Mitchell, 2012; Zettlemoyer and Collins, 2005, 2007). Additional information may also be used to generate lexicon entries, such as the names of entities in the knowledge base.

Applying the above heuristics to the "cities in Texas" example results in the following lexicon Λ:

$$\begin{array}{lllll}
\text{cities} & := & N & : & \lambda x.\textsc{city}(x) \\
\text{cities} & := & N & : & \lambda x.x = \textsc{Texas} \\
\text{Texas} & := & N & : & \lambda x.\textsc{city}(x) \\
\text{Texas} & := & N & : & \lambda x.x = \textsc{Texas} \\
\text{in} & := & (N\backslash N)/N & : & \lambda f.\lambda g.\lambda x.\exists y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x, y)
\end{array}$$

The generated lexicon includes the desired entries but also includes some implausible entries, such as the second entry for "cities." It is acceptable to generate these entries because the training process will learn that they are unlikely to appear in correct parses. However, these implausible entries make parsing less efficient by increasing the size of the grammar, and they may also reduce the generalization ability of the parser. Therefore, it is undesirable to include many such entries in the lexicon. Several improved algorithms for inducing a lexicon are described in Section 3.1.3; these algorithms aim to jointly learn both the lexicon and the semantic parsing model.

The combinators C allowed by the grammar can simply be manually selected from the set of possibilities permitted by CCG. For example, the parser can use functional application and composition, but not type-raising. Heuristic rules can also be used to define additional combinators; Chapter 4 uses this approach to define a set of binary combinators for parsing noun compounds.

2.2.3 Semantic Parsing Model

Given a grammar G, the next step in the algorithm is to define a structured prediction model Γ for the semantic parser. This model is used to disambiguate between the many logical forms for a sentence permitted by the grammar. Given a sentence s, this model produces a score for a logical form ℓ and syntactic CCG parse tree t according to:

$$\Gamma(\ell, t, s; \theta, G) \quad = \quad \theta^T \phi(\ell, t, s)$$

Above, the tuple (ℓ, t) represents a semantic parse of the sentence. It has two components because multiple syntactic parse trees t can result in the same logical form ℓ and vice versa. The feature function ϕ produces a feature vector for a semantic parse. These features are typically counts of various substructures that occur in the parse (see below). The model parameters θ contain a weight for each feature, and the score of a semantic parse (ℓ, t) for the sentence s is the inner product of the parameter and feature vectors. Although not represented in the above equation, the grammar G defines the set of possible semantic parses (ℓ, t) that can be scored in this fashion. Throughout this thesis, semantic parsers are trained using discriminative non-probabilistic linear models of the form above. However, probabilistic loglinear models can also be used for semantic parsing (e.g., (Zettlemoyer and Collins, 2005)) using very similar training algorithms.

Features

The feature function ϕ determines the parameterization of the semantic parsing model. In order to enable efficient inference with CKY, these features must factorize according to the structure of the parse tree. Given a sentence s and a semantic parse (ℓ, t), let $(w := X : l) \in \textsc{lex}(\ell, t, s)$ be the set of lexicon entries in the parse, and let $(X \;\; Y \implies Z) \in \textsc{comb}(\ell, t, s)$ be the set of instantiated combinators. The feature function ϕ is the sum of distinct lexicon and combinator feature functions, applied to each of these elements:

$$\phi(\ell, t, s) = \sum_{(w,X,l) \in \textsc{lex}(\ell,t,s)} \phi_{lex}(w, X, l) + \sum_{(X,Y,Z) \in \textsc{comb}(\ell,t,s)} \phi_{comb}(X, Y, Z)$$

The lexicon and combinator feature functions are typically composed of many indicator functions, each of which returns the value 1 for a particular set of inputs and 0 otherwise. For example, a lexicon feature function may return 1 only if $w =$ city, $X = N$ and $\ell = \lambda x.\textsc{city}(x)$, effectively detecting if the lexicon entry is town $:= N : \lambda x.\textsc{city}(x)$. An example combinator feature function may return 1 only if the combinator is the functional application $N/N \;\; N \implies N$. ϕ_{lex} and ϕ_{comb} both return vectors where each entry in the vector is the value of one such feature function. $\phi(\ell, t, s)$ is produced by summing these vectors across all lexicon entries and instantiated combinators; each entry of the resulting vector is the number of occurrences of a particular feature in the entire parse.

Many additional features can be defined that factorize in this manner. For example, if the sentence is part-of-speech tagged, the lexicon features can include an indicator feature for POS-tag/syntactic category pairs. Or, if the grammar is annotated with head-passing rules, the combinator features can include the head words of the combined spans.

Inference

Inference in this model corresponds to predicting the most likely semantic parse for a sentence s:

$$(\ell^*, t^*) \quad \leftarrow \quad \arg\max_{(\ell,t)} \theta^T \phi(\ell, t, s)$$

Note that this maximization is defined over the set of semantic parses permitted by the grammar G, though this dependence is left implicit. Assuming that ϕ factorizes according to the structure of the parse tree, this maximization can be solved exactly using the CKY algorithm (with the caveats described in Section 2.1.5).

2.2.4 Training

Training the semantic parser amounts to selecting the model parameters θ using a data set of sentence/logical form pairs, $\{(s_i, \ell_i)\}_{i=1}^n$. The structured perceptron algorithm is a simple way to select θ (Collins, 2002; Zettlemoyer and Collins, 2007). This algorithm can be viewed as minimizing the following objective function:

$$O(\theta) \quad = \quad \sum_{i=1}^{n} | \max_{\hat{\ell},\hat{t}} \Gamma(\hat{\ell}, \hat{t}, s_i; \theta, G) - \max_{t^*} \Gamma(\ell_i, t^*, s_i; \theta, G)|_+$$

This objective function consists of two terms: the first term is the score of the highest-scoring semantic parse of s_i under the current model and the second term is the score of the highest-scoring semantic parse of s_i with the correct logical form, ℓ_i. $| \cdot |_+$ denotes the positive part of the expression. Thus, the objective value for an example is nonzero only if the highest-scoring semantic parse has an incorrect logical form.

This objective function can be minimized using stochastic subgradient descent. This algorithm initializes the parameter vector to zero, then iterates over the examples in the data set performing a parameter update for each one. Let θ^j denote the parameter vector produced by the jth iterative update, with $\theta^0 = 0$. Given the example (s_i, ℓ_i), the gradient update is:

$$\begin{aligned}
\hat{\ell}, \hat{t} &\leftarrow \arg\max_{(\ell,t)} \theta^T \phi(\ell, t, s_i) \\
t^* &\leftarrow \arg\max_{t} \theta^T \phi(\ell_i, t, s_i) \\
\theta^{j+1} &\leftarrow \theta^j + \phi(\ell_i, t^*, s_i) - \phi(\hat{\ell}, \hat{t}, s_i)
\end{aligned}$$

This update first solves two inference problems. The first inference problem is simply to predict the best semantic parse $(\hat{\ell}, \hat{t})$ of s_i given the current model parameters θ^j. The second inference problem is to predict the best semantic parse that results in the correct logical form, ℓ_i. This second problem may require an approximate inference algorithm – see Section 2.2.5 for details. Finally, the parameter update adds the features of the best correct parse to θ while subtracting the features of the best predicted parse. Note that if the predicted parse is the same as the correct parse, the parameters do not change.

The structured perceptron algorithm is not guaranteed to converge to a parameter value, so training is performed by running a fixed number m of updates. The algorithm returns either the final parameter vector, θ^m, or the average of the parameters across iterations, $\frac{1}{m}\sum_{i=1}^{m} \theta^i$. Empirically, the average of the parameters results in better performance (Collins, 2002).

2.2.5 Additional Considerations

This section describes some common technical issues encountered when building semantic parsers.

Out-of-Vocabulary Words

At test time, a semantic parser is likely to encounter sentences containing words that did not appear in the training data. As described, the semantic parser has no mechanism for handling these out-of-vocabulary words, and therefore it will not be able to parse these sentences.

There are two common solutions to this problem. One solution is to create default lexicon entries for out-of-vocabulary words. These entries can be created by mapping all out-of-vocabulary words to a special symbol, UNK, then including entries for UNK in the lexicon. This approach is typically employed by wide-coverage syntactic parsers (Clark and Curran, 2007a).

If a complete syntactic parse is not necessary, another solution is to allow the parser to skip words while parsing. Word skipping can be implemented by adding a special SKIP nonterminal symbol to the grammar, allowing this nonterminal to be produced by every word, and allowing it to combine with other nonterminals without changing the nonterminal. A parser with word skipping will produce partial parse trees containing only a subset of the tokens in a sentence, i.e., the tokens that are not mapped to SKIP in the parse tree. This solution is often used in natural language question answering applications.

Richer Syntactic Categories

CCG syntactic categories can be annotated with two kinds of additional information to produce richer syntactic analyses. First, basic categories (e.g., S and NP) may have associated syntactic features given in square brackets. These features are used in CCGbank (Hockenmaier and Steedman, 2002a) to distinguish variants of basic syntactic categories, e.g., $S[\text{dcl}]$ indicates a declarative sentence. These syntactic features can also be associated with variables whose values are filled during parsing via unification. For example the adverb category $(S[X]\backslash N)/(S[X]\backslash N)$ contains the variable $[X]$. Applying this category to a declarative, intransitive verb $(S[\text{dcl}]\backslash N)$ returns $(S[\text{dcl}]\backslash N)$, as the variable $[X]$ is unified with $[\text{dcl}]$. These variables reduce the number of syntactic categories required to model these features; in this case, a single adverb category suffices even though intransitive verbs have multiple possible feature values.

Second, each category may be annotated with head passing and dependency information, indicated using subscripts. For example, the preposition "in" may have the category $(N_1\backslash N_1)/N_2$.

Parsing the phrase “town in Texas” with this category creates two predicate argument dependencies, from “in” to “town” and “Texas”, respectively. The head word of the phrase is “town” due to the coindexing of the argument and return categories in $(N_1\backslash N_1)$. Predicate-argument dependencies are used to evaluate parser performance against CCGbank and to generate parse tree features for training a parsing model.

Approximate Inference

Approximate inference is usually necessary to train a CCG semantic parser. The difficult inference problem is, given a sentence and logical form (s_i, ℓ_i), to find the highest-scoring syntactic derivation t^* that produces ℓ_i:

$$t^* \quad \leftarrow \quad \arg\max_t \theta^T \phi(\ell_i, t, s_i)$$

This problem is difficult because the relationship between the derivation t and logical form ℓ_i is complex. It is therefore unclear how to enforce local restrictions during parsing – that is, on individual entries in the CKY chart – to guarantee that all syntactic derivations produce ℓ_i. Consequently, this problem cannot be efficiently solved with CKY.

One approach to solving this inference problem is to implement a CKY-style parsing algorithm that maintains both a syntactic category X and logical form ℓ in each chart entry. This algorithm is inefficient because the number of possible logical forms for a span can grow exponentially in the length of the sentence, and hence parsing can take exponential time. Theoretically, this approach can find the exact maximizer t^*. In practice, however, the grammar is usually too large to permit an exact search, so beam search is used to produce an approximate result. Given a sentence, beam search generates a ranked list of semantic parses (t_j, ℓ_j); if one of these has the correct logical form, $\ell_j = \ell_i$, then t_j is taken as the approximate maximizer.

2.3 Discussion

This chapter has presented a simple algorithm for building a semantic parser with Combinatory Categorial Grammar (CCG). The algorithm has two steps. First, it defines a grammar for the parser using a collection of heuristic lexicon induction rules. This grammar permits many possible semantic parses for a sentence. Second, it trains a structured prediction model to predict which of these semantic parses is correct. This step uses a linear model trained with the structured perceptron algorithm.

The presented algorithm for training a semantic parser has many shortcomings. Three significant limitations are (1) the heuristic lexicon induction rules, (2) the training data requirement of sentences with annotated logical forms and (3) the knowledge base must be known *a priori*. Many solutions have been proposed for these problems; these are discussed in the following chapter.

Chapter 3

Related Work

This thesis is related to work from many different areas of natural language processing. Section 3.1 describes related work on semantic parsing, which is used throughout this thesis. Section 3.2 describes work on broad coverage syntactic parsing with Combinatory Categorial Grammar. A syntactic CCG parser is trained in Chapter 4 and used in Chapter 5. Section 3.3 describes related work on information extraction, which inspires the distantly-supervised training regime in Chapter 4 and the probabilistic matrix factorization technique in Chapter 5. Finally, Section 3.4 covers work on jointly understanding natural language and perceiving the world, which is the problem considered in Chapter 6.

3.1 Semantic Parsing

Semantic parsing is the problem of mapping natural language text to a formal representation of its meaning, known as a *logical form*. In this thesis, a logical form is a statement in lambda calculus; however, alternative formalisms have been used in prior work. In every formalism, a logical form is a program that can be automatically evaluated by a computer against a world representation (typically a logical knowledge base). Because the logical form can be evaluated, it has a well-defined formal semantics.

Many distinct formalisms and supervision assumptions have been proposed to train semantic parsers. All of the work in this section requires a logical knowledge base to be given as input. Work that uses a learned knowledge base is deferred to Section 3.4.

3.1.1 Formalisms

Many formalisms have been proposed for semantic parsing. Although superficially different, most of these formalisms are very similar in that the semantic parser is defined by a grammar and a machine-learned scoring model. The grammar defines the space of possible parses for each sentence, and the scoring model is used to select a single parse from these possibilities. Thus, at a high-level these techniques are similar to the basic approach described in Chapter 2.

The first approaches to semantic parsing were based on inductive logic programming (ILP). ILP induces a logic program (e.g., in Prolog) from a collection of positive and negative examples such that the program entails all of the positive examples and none of the negative examples. The CHILL system induces a shift-reduce grammar from a corpus of sentences with annotated logical forms, then learns control rules for this grammar with ILP (Zelle and Mooney, 1993, 1996). The ILP algorithm learns when each induced grammar rule should be applied, essentially performing the role of the scoring model. On GEOQUERY, this approach achieves an accuracy of 68%. The COCKTAIL system improves on the ILP heuristics used by CHILL to achieve an accuracy of 79.4% (Tang and Mooney, 2001). SILT is a similar system whose grammar consists of transformation rules from strings or parse trees to logical form. A heuristic scoring function determines which rules to apply (Kate et al., 2005). This system performs somewhat worse than COCKTAIL, with an accuracy around 65%.

Statistical techniques for parsing context-free grammars have also been used for syntactic parsing. The SCISSOR system learns a context-free grammar whose nonterminals encode both syntax and semantics (Ge and Mooney, 2005). This system learns a generative model over these parse trees based on Collins model 2 (Collins, 1997) using fully annotated syntactic and semantic parse trees for training. SCISSOR is shown to compare favorably to prior work on GEOQUERY and the ROBOCUP coaching data set. A similar, but unlearned, model has been used for direction following (MacMahon et al., 2006). Follow-up work pipelines this model, first syntactically parsing the text with a syntactic parser then performing a semantic analysis according to the structure of the syntactic parse tree (Ge and Mooney, 2009). Experimental results with this pipelined model demonstrate that high syntactic parsing accuracy is critical to achieve high semantic parsing performance. Another similar context-free grammar approach is to generate a hybrid tree that contains both meaning representation and word nodes (Lu et al., 2008). This work also trains a generative parsing model inspired by Collins, but uses Expectation Maximization to avoid requiring annotated syntactic trees. When combined with a discriminative reranker, this approach achieves good results. KRISP is a another semantic parsing system that induces a context-free grammar over the text and logical form; however, this system uses SVMs with string kernels to score the production rules invoked for each sentence span (Kate and Mooney, 2006). A final variant of this statistical parsing approach is implemented in WASP, which uses probabilistic synchronous context-free grammars (PSCFG) (Wong and Mooney, 2006, 2007). PSCFG is a variant of context free grammar that simultaneously generates two strings instead of one (Wu, 1997). For semantic parsing, one side of the PSCFG generates natural language, and the other side generates logical forms.

The previously mentioned techniques all construct semantic parsers using phrase-structure grammars. DCS is a semantic parsing formalism that uses a dependency grammar (Berant et al., 2013; Liang et al., 2011, 2013). In DCS, words invoke entities and predicates, and dependency edges connect arguments of these predicates through join operations. Additional operations are available to handle divergence of syntactic and semantic scope; these operations are used to model quantifiers. Some other work has also explored dependency-based semantic parsing via a conversion from dependency to CCG parses (Cantrell, 2009).

Combinatory Categorial Grammar (CCG) has also been used extensively to develop semantic parsers (Artzi and Zettlemoyer, 2013; Artzi et al., 2014; Cai and Yates, 2013a,b; Krishnamurthy and Kollar, 2013; Krishnamurthy and Mitchell, 2012, 2014; Kwiatkowski et al., 2010, 2011, 2013; Lee et al., 2014; Matuszek et al., 2012; Reddy et al., 2014b; Zettlemoyer and Collins, 2005, 2007). Chapter 2 explains this formalism in detail. CCG was introduced for semantic parsing by Zettlemoyer and Collins (Zettlemoyer and Collins, 2005). Much of the follow-up work has focused on subproblems within semantic parsing, such as inducing the lexicon or using less supervision. This work is described in more detail later in this section. All of the semantic parsers in this thesis are built using CCG.

A limitation of the above parsing approaches is that the induced grammars can be overly restrictive, disallowing potentially valid semantic parses. An alternative line of work has explored generating and scoring all possible logical forms for a given sentence (Clarke et al., 2010; Goldwasser et al., 2011; Roth and Yih, 2007). This work formulates inference as an integer linear program that maximizes the sum of an alignment score between words and predicates and a predicate-argument score. In essence, this work uses a very permissive grammar formalism, where words are mapped to predicates that can then be composed in any (type-consistent) manner.

Finally, some work has aimed to formulate semantic parsing as instances of other NLP tasks. Logical forms can be linearized into a string representation, which makes machine translation a natural fit (Andreas et al., 2013). This approach achieves competitive results to state-of-the-art systems on GeoQuery while requiring very little additional engineering effort. Another approach is to formulate semantic parsing as a paraphrasing problem (Berant and Liang, 2014). Here, given input language, a number of candidate logical forms are generated. Canonical textual representations of these logical forms are generated using templates, then a paraphrase model scores the match between these representations and the input language. This model was designed to improve performance on large knowledge bases, where it is difficult to resolve the mapping from language to KB symbols.

Given the diversity of grammar formalisms, it is natural to ask which is the best. However, on small data sets such as GeoQuery, the various formalisms perform roughly comparably. Considering the similarities between these formalisms, the performance differences between these approaches are more likely the consequence of other design choices, such as of a discriminative or generative model, rather than the formalism itself. The large semantic parsing data sets available today have less compositional language and performance is largely determined by the parser's ability to match words to KB symbols (Berant and Liang, 2014; Kwiatkowski et al., 2013). Consequently, these results also do not demonstrate a clear advantage for any formalism.

3.1.2 Types of Supervision

Many forms of supervision have been explored in previous work, spanning labeled logical forms to essentially no supervision whatsoever. This section covers each type of supervision, roughly ordered by most to least annotation effort.

Logical Forms

Most of the early work on semantic parsing trained parsers on a corpus of sentences with annotated logical forms (Kate and Mooney, 2006; Kate et al., 2005; Kwiatkowski et al., 2010; Lu et al., 2008; Wong and Mooney, 2006, 2007; Wu, 1997; Zelle and Mooney, 1993, 1996; Zettlemoyer and Collins, 2005, 2007). In this setting, the training data consists of a knowledge base K and a collection $\{(s_i, \ell_i)\}_{i=1}^{n}$ of sentences s_i with annotated logical forms ℓ_i. A basic approach to training a semantic parser with this form of supervision was described in Chapter 2.

This form of supervision, while effective, has fallen out of favor recently due to the perceived difficulty of annotating logical forms. However, annotated logical forms do have some advantages over other forms of supervision. Specifically, they are helpful for lexicon induction, because an alignment between text and logical form fragments can be computed. These lexicon induction techniques are discussed in Section 3.1.3.

Question/denotation pairs

Another form of supervision is question/denotation pairs. A denotation is the value produced by evaluating a logical form against a knowledge base; in question answering, the denotation is the answer to the question. There are two common tasks that use this form of supervision. The first is answering natural language questions using a database (as above) where that the semantic parser is trained from question/answer pairs. This task was introduced by Liang et al. (Liang et al., 2011), and has been used in several more recent works on semantic parsing (Berant and Liang, 2014; Berant et al., 2013). The second task is learning to interpret natural language directions to navigate a robot through a virtual environment (Chen and Mooney, 2011; Matuszek et al., 2010; Shimizu and Haas, 2009). The supervision provided in this task is natural language directions paired with the correct path through the environment. A superficially different setting is training a semantic parser given a feedback oracle that provides a binary correct/incorrect signal (Clarke et al., 2010); however, since the oracle can be queried at will, the same algorithms for learning from denotations function in this setting.

These tasks share a formalization. The training data consists of a knowledge base K and a collection $\{(s_i, \gamma_i)\}_{i=1}^n$ of questions s_i with annotated denotations γ_i. In question answering, the knowledge base is the database and the denotation is a set of entities that is the correct answer to the question. In direction following, the knowledge base is a logical representation of the robot's environment, and the denotation is the correct path. (Presumably, the semantic parser for direction following includes an action type that evaluates to paths in the environment.) The semantic parser is trained to predict a logical form ℓ_i^* for s_i such that evaluating ℓ_i^* on K produces the annotated denotation, i.e., $\textsc{eval}(\ell_i, K) = \gamma_i$. An alternative view of this form of supervision is that there is a set of correct logical forms for each sentence, specifically the set $\{\ell : \textsc{eval}(\ell, K) = \gamma_i\}$.

There are three high-level algorithms for training semantic parsers from this kind of supervision. The first is to take a stagewise approach. This approach first infers a logical form ℓ_i for each training example s_i with the correct denotation. These logical forms are then presumed to be correct, and therefore can used to train a semantic parser from sentence/logical form pairs. Naturally, inducing the logical forms is the main challenge. This approach has been used in the direction following setting, where it is straightforward to induce logical forms for simple paths (e.g., the path for "go straight") (Chen and Mooney, 2011). However, generating more complex logical forms requires an additional mechanism to score language-logical form pairs.

The second approach is an Expectation-Maximization (EM) approach (Dempster et al., 1977) that iteratively performs the following two steps. The E-step predicts a distribution over logical forms with the correct denotation for each sentence using the current semantic parser parameters. The M-step then retrains the semantic parser using the predictions of the previous step as weighted labels. Kim and Mooney use such an EM approach to train a context-free grammar semantic parser (Kim and Mooney, 2012). A discriminatively-reranked version of this algorithm achieves state-of-the-art results on a simulated direction following data set (Kim and Mooney, 2013). The KRISPER (Kate and Mooney, 2007) and WASPER (Chen and Mooney, 2008) systems can also be viewed as using non-probabilistic variants of EM.

The third, and most direct approach, is a stochastic gradient approach similar to the one presented in Section 2.2.4. The difference is that, in this setting, the single correct logical form for a sentence is replaced by the set of logical forms with the correct denotation. This modification requires maximizing or marginalizing over this set during training, depending on the choice of objective function. However, both of these operations can be performed approximately by using a beam search and evaluating the resulting logical forms on the knowledge base. This technique is used by all of the work on question answering (Berant and Liang, 2014; Berant et al., 2013; Liang et al., 2011). A downside of this approach is that logical forms must be evaluated against the knowledge base within the gradient computation, which can be inefficient.

An advantage of question/denotation pairs is that the task is formalism agnostic: the parser does not have to output a logical form, just the correct answer. Furthermore, experimental results suggest that semantic parsers trained with question/answer pairs have comparable performance to parsers trained with logical forms (Liang et al., 2011).[1] However, question/denotation pairs are still expensive to annotate, though less expensive than logical forms.

Grounded language acquisition

Grounded language acquisition is the problem of learning the mapping between language and an environment. As a motivating example of this problem, consider an agent watching a baseball game and listening to the announcer's commentary. In this example, the agent perceives both language (the announcer's statement) and the state of the world (the events in the baseball game). Although the meaning of the language is not directly observed – it has no annotated semantics or answer – there is clearly useful semantic information in the correlation between language and the environment. Grounded language acquisition is the problem of learning to understand language using this kind of supervision.

Grounded language acquisition can be formulated as a semantic parsing problem with an indirect form of supervision. Here, the training data $\{(s_i, K_i)\}_{i=1}^{n}$ consists of sentences s_i paired with formal representations of the environment K_i (i.e., knowledge bases). The object is to predict a logical form ℓ_i for the sentence s_i such that $\textsc{eval}(\ell_i, K_i) = \textsc{true}$. The training signal for the parser is the assumption that every sentence expresses a true fact about the environment, which trains the parser to correlate language with the environment. Note that, since logical forms are presumed to evaluate to TRUE, this setting is very similar learning from denotations, as above.

Much of the work on grounded language acquisition has been on the task of generating natural language sportscasts of RoboCup soccer games. This task is very simple because the correct

[1] It seems likely that, for each question, only a few possible logical forms evaluate to the correct answer. That is, $\{\ell : \textsc{eval}(\ell, K) = \gamma_i\}$ is small. This claim is more likely given the following additional restrictions on logical forms: (1) questions typically reference an entity that should appear in the logical form; (2) the size of the logical form is not too large; and (3) answers are sets of entities. If this claim is true, it is unsurprising that question-answer pairs work well as there is little added ambiguity during training.

logical form ℓ_i for each sentence s_i is a single relation instance. Chen and Mooney (Chen and Mooney, 2008) study this problem and use KRISPER (Kate and Mooney, 2007) to train the semantic parser. Follow up work extends this approach to a fully generative model trained with EM (Kim and Mooney, 2010); this model adds an event selection component to an existing generative semantic parsing model (Lu et al., 2008). This method is demonstrated to improve upon the matching between natural language and logical forms. This model was further improved by being formulated as a single context-free grammar (Börschinger et al., 2011). However, this grammar includes logical form fragments in the nonterminals, such as specific relation instances; this embedding limits the approach to simple logical forms. An extension of this context-free grammar approach has been used for direction following (Kim and Mooney, 2012); however, even this approach cannot generate logical forms that are more complex than any logical form seen during training.

The terminology surrounding "grounded language acquisition" is unfortunately ambiguous. This term can refer to the problem setting described here, but has also been used to refer to training semantic parsers from denotations, as above. It can also refer to work that interprets language with respect to non-logical physical environments, e.g., images or actions of a robot (see Section 3.4). In the remainder of this thesis, I reserve the term "grounded" to refer to this third problem setting, where language is understood with respect to an environment and *no logical knowledge representation of this environment is provided to the understanding system.* This caveat is crucial, as otherwise all work on semantic parsing is "grounded" in a knowledge base, rendering the term meaningless.

Distant Supervision

Semantic parsers have also been trained with distant supervision. In this setting, the input to training is a corpus of unlabeled sentences $\{s_i\}_{i=1}^n$ and a large knowledge base K, such as NELL (Carlson et al., 2010) or Freebase (Bollacker et al., 2008). This line of work is inspired by recent research that has demonstrated that accurate sentential relation extractors can be trained using distant supervision (Bunescu and Mooney, 2007; Craven and Kumlien, 1999; Hoffmann et al., 2011; Mintz et al., 2009; Riedel et al., 2010; Wu and Weld, 2010). These relation extraction systems can extract hundreds of predicates, but lack a compositional semantic representation.

Note that the distant supervision setting is closely related to the grounded language acquisition setting above. The difference in the distant supervision setting is that only one large knowledge base is provided to the system, as opposed to multiple, smaller knowledge bases. Furthermore, the unlabeled sentences are less likely to refer to predicate instances in the knowledge base, as the sentences are drawn from the web as opposed to generated in the context of a particular environment. Consequently, the distant supervision setting has much more ambiguity in the mapping from language to the knowledge base.

Training algorithms vary in the distant supervision setting, as many different assumptions can be made about the relationship between the language and the knowledge base. However, the general structure of these algorithms is similar: they find sentences in the corpus that mention entities (e_1, e_2) in the knowledge base, then assume that (some of) these sentences express relation instances $r(e_1, e_2)$ from the knowledge base. This assumption provides the training signal for the semantic parser.

Cai and Yates (Cai and Yates, 2013a,b) train a semantic parser by generating a data set of sentences with labeled logical forms from the distant supervision inputs. This approach searches the corpus for sentences that mention entity pairs that participate in a particular relation, then uses an additional assessment stage to prune sentences that are unlikely to express the relation. These sentences are then automatically labeled with a logical form for the relation. This data is then fed into the UBL algorithm (Kwiatkowski et al., 2011) to train a semantic parser.

Reddy et al. (Reddy et al., 2014b) present a distantly-supervised graph-based approach to semantic parsing. Unlike other approaches, this approach produces a graph-structured semantic parse by directly transforming the output of a CCG syntactic parser. The supervision for this system is created by replacing entity mentions in sentences by variables, then training the semantic parser to predict a denotation for this variable that includes the mentioned entity. (In some cases, even stronger constraints can be applied. For example, if the mention is preceded by the word "the," then the correct denotation includes exactly one entity.) This system achieves state-of-the-art results on both FREE917 and WEBQUESTIONS.

Chapter 4 also presents a distantly-supervised algorithm for training a semantic parser (Krishnamurthy and Mitchell, 2012, 2014). This algorithm assumes that at least one sentence mentioning (e_1, e_2) expresses any relation r such that $r(e_1, e_2)$. This assumption has been successfully used for distantly supervised relation extraction (Hoffmann et al., 2011), where it has been shown to outperform more naïve labeling assumptions.

Unsupervised

Some work has trained semantic parsers in an unsupervised fashion, that is, without any per-sentence semantic annotations or reference to relation instances in a knowledge base. These parsers are trained using a corpus of sentences and the entity names in a knowledge base.

Goldwasser et al. (Goldwasser et al., 2011) present an EM-like algorithm for training a semantic parser using only a corpus of sentences. The algorithm is similar to EM, except that an alternative set of confidence measures are used to determine which model predictions to retrain on. The system is evaluated on the GeoQuery data set and manages to achieve an accuracy of 66%. The evaluation suggests that the confidence measures are not very important – self-training, i.e., retraining the semantic parser on all of its predictions on each iteration, achieves an accuracy of 65%.

Grounded unsupervised semantic parsing (GUSP) (Poon, 2013) is another unsupervised approach. This system uses a dependency parse of the question as an input, assigning it a semantics that factors according to the same parse structure (with some special operations to handle divergence between syntax and semantics). The EM algorithm is used for training. On the ATIS data set, GUSP achieves an accuracy of 84%, which is comparable with state-of-the-art supervised systems.

It is somewhat surprising that these unsupervised approaches are so effective. A partial explanation for these results is that both approaches use intelligently initialized mappings from words to knowledge base predicates. Both systems use entity and predicate names from the knowledge base as well as similar names drawn from outside lexical resources, such as WordNet (Miller, 1995) and DASH (Pantel et al., 2009). Restrictive type constraints and relatively small knowledge bases also reduce the ambiguity faced by these systems. Larger knowledge bases will make unsupervised learning more difficult; it would be interesting to see if such an unsupervised approach can work on a more complex data set, such as Free917.

3.1.3 Lexicon Induction

A lexicon is a mapping from words to syntactic categories and logical forms that is the basis of the grammar of a CCG semantic parser (see Section 2.1.3). *Lexicon induction* is the problem of learning a lexicon from a data set. Although manual heuristics can be used for this problem, they are time-consuming to design and may lead to a bloated lexicon. It is therefore desirable to automatically learn the contents of the lexicon. Note that this problem is potentially difficult, as it is a form of structure learning. The lexicon induction problem is typically considered in the annotated logical form setting.

Zettlemoyer and Collins present two algorithms that combine a collection of manual heuristics with a semantic parser training procedure that adds and removes lexicon entries (Zettlemoyer and Collins, 2005, 2007). Here, the heuristics generate a large number of potential entries, but only a few are included in the final lexicon by the training procedure. These algorithms differ in that the first uses batch lexicon updates while the other uses online updates.

Kwiatkowski et al. (Kwiatkowski et al., 2010) present a distinct approach to lexicon induction based on splitting logical forms. The lexicon is initialized with a single lexicon entry per training example. The logical form for this entry is then split using restricted higher-order unification; given the logical form ℓ, this process finds logical forms f, g such that $f(g) = \ell$ or $\lambda x.f(g(x)) = \ell$. The splits are made within a stochastic gradient procedure, considering only lexicon entries appearing in the highest scoring parse of a training sentence.

Artzi et al. (Artzi et al., 2014) present a batch lexicon induction algorithm that focuses on reducing the size of the learned lexicon. The batch approach enables the algorithm to aggregate statistics about the number of times each lexicon entry is used to parse a training example, which help the algorithm select the entries to include in the lexicon. This approach is shown to produce smaller lexicons with approximately equivalent single-sentence semantic parsing performance.

Notably, several approaches to lexicon induction rely on word alignment algorithms from machine translation. These word alignment algorithms are based on simple string-to-string translation models and are designed to learn word-to-word translation probabilities. When applied to sentence/logical form pairs, they learn word-to-predicate translation probabilities. The GIZA++ word alignment system (Och and Ney, 2003) has been used to generate the lexicon (Ge and Mooney, 2009) and initial feature scores (Kwiatkowski et al., 2010). In both cases, the word alignment helps guide the search over lexicons toward promising lexicon entries.

3.2 Wide-Coverage Syntactic Parsing with CCG

Wide-coverage syntactic CCG parsers aim to produce syntactic CCG analyses of any sentence in a language. Wide-coverage parsing is more difficult than semantic parsing in restricted domains because it requires a larger vocabulary of words and processing more complex syntactic structures. Thus, wide-coverage parsers use sophisticated parsing models with considerably richer features than semantic parsers. Efficiency is also a major concern, and much work has gone in to developing faster CCG parsing algorithms.

CCGbank is the standard data set for training and evaluating syntactic CCG parsers (Hockenmaier and Steedman, 2002a). This data set is an automatic conversion of the Penn Treebank

(Marcus et al., 1993) into the CCG formalism. In addition to annotated syntactic trees, this corpus contains predicate-argument dependency structures between head words and their arguments; these dependencies are similar to those produced by dependency parsing, except that they additionally capture head passing and conjunctions. Precision and recall of dependency structures is typically used to evaluate syntactic CCG parsers. CCGbank is split into sections, with sections 02-21 used for training (39,604 sentences), section 00 for validation (1,913 sentences), and section 23 for evaluation (2,407 sentences).

3.2.1 Wide-Coverage Parsers

The first wide-coverage syntactic CCG parsers used generative parsing models. Hockenmaier and Steedman (Hockenmaier and Steedman, 2002b) present a model that generates the syntactic CCG tree in a similar manner as generative context-free grammar parsers (Collins, 1997). A limitation of this approach is that it does not model long-range dependencies. Clark et al. (Clark et al., 2002) present a conditional model that exclusively generates the sentence's dependency structures. However, this model is deficient in that the probabilities of all CCG parses for a sentence do not sum to one. Hockenmaier (Hockenmaier, 2003a) unifies both approaches with a properly-normalized generative model of both CCG syntactic trees and dependency structures. Generative models have fallen out of favor for syntactic parsing because – unlike discriminative models – they do not permit rich, overlapping sets of features.

More recent parsers use discriminative parsing models. The C&C parser uses a discriminatively-trained log-linear model to score CCG parses, using a rich set of both syntactic and dependency features (Clark and Curran, 2007a). This parser is available as a software package and is commonly used in applications. Follow-up work has trained this parser with the structured perceptron algorithm (Clark and Curran, 2007b). The features of the syntactic parsing model in Chapter 4 are based on the C&C parser.

Fowler and Penn observe that a context-free subset of CCG is sufficient to model the CCGbank grammar (Fowler and Penn, 2010). Based on this observation, they apply the Petrov and Klein CFG parser (Petrov and Klein, 2007) for CCG parsing. This parser is a generative model that automatically splits and merges syntactic categories to better model the observed syntactic trees. This approach outperforms C&C.

Auli and Lopez (Auli and Lopez, 2011a) train a CCG parser using a softmax-margin objective that aims to maximize dependency F-measure instead of conditional loglikelihood. This work is motivated by the observation that the training loss should be similar to the test loss to ensure the best performance (Och, 2003). A discriminative CCG parsing model with integrated supertagger features (Auli and Lopez, 2011c), trained using this approach, achieves state-of-the-art CCG parsing results on CCGbank.

3.2.2 Faster Parsing Algorithms

Supertagging is widely recognized as critical for producing efficient CCG parsers (Bangalore and Joshi, 1999; Clark and Curran, 2004). A supertagger prunes the set of possible syntactic categories for each token in a sentence before parsing, and subsequent parsing is restricted to the assigned syntactic categories.

Some recent work has developed alternative parsing algorithms that are more efficient than CKY. One approach is A* parsing (Klein and Manning, 2003), which uses A* search to intelligently select entries of the CKY chart to expand. Initial work on A* parsing with CCG did not show dramatic speed improvements (Auli and Lopez, 2011b), but recent work has developed novel heuristics for the search that lead to a threefold speedup over CKY (Lewis and Steedman, 2014).

Another approach is shift-reduce parsing (Xu et al., 2014; Zhang and Clark, 2011), which performs a beam search over parses while processing the sentence in left-to-right order. Although this approach has yet to demonstrate significant speedups in CCG, shift-reduce parsing has been shown to improve the speed of dependency parsers (Nivre et al., 2006), suggesting the promise of this approach. Another interesting aspect of shift-reduce parsing is that it does not require the parser's features to factorize according to local operations in the parse tree.

3.2.3 Semantics

Some work has also attempted to automatically derive logical semantic representations directly from syntactic CCG parses. Boxer (Bos, 2005, 2008; Bos et al., 2004) is a rule-based system that produces discourse representation structures (Kamp and Reyle, 1993) from the output of the C&C parser. This system assigns a semantic representation to each token of a sentence on the basis of its syntactic category, part-of-speech tag, and the word itself, then composes these representations according to the structure of the CCG parse. The predicates in the output semantic representation are derived directly from the words in the input text.

Lewis and Steedman (Lewis and Steedman, 2013) present a semantic analysis that combines a rule-based system with distributional clustering of predicates. This system similarly uses a collection of rules to assign a semantics to each token in a sentence, with predicates derived from the words themselves. However, these per-word predicates are then clustered using distributional similarity. This clustering process identifies synonyms, producing a more language-independent semantic representation.

Note that these approaches do not produce a semantic representation tied to a knowledge base. Consequently, these approaches cannot be directly used for the semantic parsing problems described above. Chapter 5 extends the above approaches by learning predicate denotations in terms of knowledge base entities. This procedure ties the semantic representation to a knowledge base, providing the same kind of semantics as semantic parsers.

3.3 Information Extraction

Information extraction systems aim to automatically populate a knowledge base by extracting predicate instances from a corpus of text. These systems vary along two significant dimensions. The first is whether the extractor is a sentential or aggregate extractor. Sentential extractors determine whether a particular sentence s expresses a relation instance $r(e_1, e_2)$, whereas aggregate extractors determine whether the instance $r(e_1, e_2)$ is expressed by one or more sentences in a corpus. Aggregate extraction is easier than sentential extraction because it can use statistical information from the entire corpus. This section focuses on sentential extractors, which are more relevant for this thesis. The second is the kind of supervision provided to train the extractor, which varies from fully labeled sentences to a handful of predicate instances and a corpus of unlabeled text.

3.3.1 Supervised

Supervised relation extraction trains sentential relation extractors using sentences that are individually annotated with expressed relation instances. Early work on relation extraction used the ACE corpus (Doddington et al., 2004). This problem can be posed as supervised multiclass classification: given a sentence and a pair of mentions, predict which relation holds between them. Various machine learning approaches have been proposed, such as SVMs (Zhou et al., 2005) and kernel methods (Bunescu and Mooney, 2005; Culotta and Sorensen, 2004; Zelenko et al., 2002). Unfortunately, the supervised learning paradigm is difficult to scale to large knowledge bases, as it is expensive to annotate a sufficient quantity of labeled data.

3.3.2 Distant Supervision

Distant supervision is a recent trend in information extraction. Distantly-supervised extractors are trained using a corpus of unlabeled text and a knowledge base, with no per-sentence relation annotations.

A key problem in this area is identifying which sentences express a given KB relation instance $r(e_1, e_2)$. One approach is to simply assume that every sentence that mentions both e_1 and e_2 expresses the relation r. Craven and Kumlien (Craven and Kumlien, 1999) use this assumption to train a sentential relation extractor using a Naïve Bayes classifier. Mintz et al. (Mintz et al., 2009) use this assumption to train an aggregate relation extractor by summing the feature vectors of each individual sentence. The Kylin system also uses this assumption to extract infobox relation instances from Wikipedia articles (Wu and Weld, 2007; Wu et al., 2008). This system tries to find each infobox instance in the text of the corresponding Wikipedia article. Later work has relaxed this constraint to identify relation instances in any Wikipedia article (Nguyen and Moschitti, 2011).

The distant supervision assumption used above is quite strong. A natural relaxation is the expressed-at-least-once assumption: at least one sentence mentioning both e_1 and e_2 expresses $r(e_1, e_2)$. Bunescu and Mooney (Bunescu and Mooney, 2007) train a string-kernel SVM relation extractor using this assumption, which they formulate as a form of multiple instance learning. Riedel et al. (Riedel et al., 2010) find that this assumption outperforms the naïve assumption. MultiR (Hoffmann et al., 2011) further improves on this assumption by modelling mutual exclusion between the multiple relations that may hold between a pair of entities. Surdeanu et al. (Surdeanu et al., 2012) extend this model by learning the function mapping individual sentence labels to aggregate labels.

A further problem with distant supervision is missing relation instances in the knowledge base. Takamatsu et al. (Takamatsu et al., 2012) address this problem by focusing on learning which textual patterns express each knowledge base relation. Min et al. (Min et al., 2013) model instances missing in the database by adding an additional latent variable to the model of Surdeanu et al. (Surdeanu et al., 2012). Xu et al. (Xu et al., 2013) propose a two-step procedure: first, a passage retrieval model is used to fill in missing instances in the knowledge base; next, MultiR is trained on the filled-in knowledge base to produce relation extractors. Ritter et al. (Ritter et al., 2013) extend MultiR with a model for missing relations that captures the difference between relation instances in the knowledge base and those expressed in text.

Chapter 4 presents an algorithm for training a semantic parser from distant supervision. This work extends the distant supervision assumptions above to work for a semantic parser that has a more expressive knowledge representation than a relation extractor.

3.3.3 Open Information Extraction

A limitation of information extraction methods is their restriction to a closed vocabulary of predicates. Closed predicate vocabularies have limited coverage, as they are typically manually defined. Furthermore, such a vocabulary may abstract away potentially relevant semantic differences, for example, by mapping both "city" and "town" to the predicate CITY. An alternative paradigm is an *open information extraction*, which assigns each natural language word or phrase its own formal predicate (Banko et al., 2007; Etzioni et al., 2005). This approach has the potential to capture subtle semantic differences and achieve high coverage.

Many systems have been built to automatically extract Open IE relation instances from text. Given a manually-specified collection of textual predicates, KnowItAll (Etzioni et al., 2005) automatically extracts predicate instances by bootstrapping from a collection of domain-independent extraction patterns (similar to Hearst patterns (Hearst, 1992)). TextRunner is a self-supervised system that extracts candidate relation instances from sentences and filters them with a Naïve Bayes classifier and via corpus-wide aggregation (Banko et al., 2007). This system was then further improved upon using a CRF sequence tagging model (Banko and Etzioni, 2008) and semantic role labeling (Christensen et al., 2010). Wu and Weld demonstrate that Wikipedia infoboxes can be used to train relation-independent extractors (Wu and Weld, 2010). ReVerb extracts open IE instances using simple part-of-speech and lexical heuristics (Fader et al., 2011). Interestingly, this approach outperforms all of the prior machine learning approaches to Open IE.

Ollie extends ReVerb to capture nominal relations by bootstrapping from ReVerb's extractions (Mausam et al., 2012).

Because Open IE instances lack canonical representations of predicates and entities, it is useful to identify synonymous predicates and entities in a corpus of Open IE extractions. DIRT is a system that clusters relations, represented as dependency paths, based on mutual information statistics of their arguments (Lin and Pantel, 2001). Resolver (Yates and Etzioni, 2007) is a system that identifies synonymous predicates and entities via a clustering procedure. This procedure clusters both entity names and predicate names on the basis of string similarity and distributional statistics.

Another approach to the redundancy problem is to use matrix factorization methods. These methods fill in missing instances in the corpus on the basis of distributional similarities between relations and entities. The matrix factorization can be viewed as performing a soft clustering. Precursors to these methods focused on the problems of finding analogies using distributional similarity (Speer et al., 2008; Turney, 2008). Universal schema combines knowledge base and Open IE relations in a single matrix (Riedel et al., 2013; Yao et al., 2013). Factoring and completing this matrix produces predictions for knowledge base relation instances on the basis of the observed open IE relations. This approach is shown to outperform state-of-the-art distantly-supervised relation extraction systems (Riedel et al., 2013). A similar matrix factorization method is used in Chapter 5 to learn the denotations of textual predicates.

3.4 Grounded Language Understanding

This section describes work on understanding the meaning of language in the context of a particular physical environment. This problem differs from semantic parsing in that the representation of the environment is not given to the system. Thus, grounded language understanding systems must learn to perceive their environment to produce a knowledge base against which language can be interpreted.

3.4.1 Understanding Visual Language

Roy (Roy, 2002) presents a model for grounding natural language object descriptions in virtual scenes of rectangles. Given natural language descriptions paired with virtual scenes with a target object, the model learns word classes, the grounded properties of each object, and syntactic information in the form of typical word orderings. This information can be used to synthesize new object descriptions, such as “the green rectangle to the right of the vertical red rectangle.”

Matuszek et al. present a model for mapping natural language object descriptions to sets of image segments (Matuszek et al., 2012). The system’s perceptual model consists of one-argument image segment classifiers that detect concepts such as “blue” and “block.” To interpret an object description, these classifiers are composed according to a CCG semantic parse of the description. A gradient descent algorithm is used to train both the image segment classifiers and the semantic parser, using a small amount of fully-labeled data and additional weakly-labeled data. The fully-labeled data is annotated with semantic parses, while the weakly-labeled data is only annotated with denotations (i.e., the set of described objects). The FUBL algorithm (Kwiatkowski et al., 2011) is run on the fully-labeled data to produce the semantic parser’s lexicon.

Yu and Siskind present a model for mapping natural language event descriptions to videos of the events (Yu and Siskind, 2013). This model uses a set of object detectors to automatically identify objects in each frame of the video. The perceptual model represents the meaning of each word as an HMM that generates object tracks in the video. To interpret an event description, these HMMs are composed into a factorial HMM based on a semantic parse of the description. The semantic parse is generated by a small, manually-defined context-free grammar.

Chapter 6 also considers the problem of understanding visual language. The model presented has a richer knowledge representation than the work above, representing both sets of objects and relations between them.

3.4.2 Direction Following

Understanding natural language directions to robots provides numerous examples of models that integrate language understanding with perceptual context. Spatial Description Clauses (SDCs) (Kollar et al., 2010) represent the semantics of commands such as "go through the double doors" using four fields: a figure (the person performing the action), a verb, a spatial relation, and a landmark. This initial work used a sequence model to produce SDCs from text, which does not permit them to recurse. The Generalized Grounding Graph (G^3) model improves on SDCs by incorporating recursion, producing a graphical model that factors according to the structure of a phrase structure parse of the language (Tellex et al., 2011). Initial work on G^3 required annotation of not only the correct action to perform, but also all referenced locations and objects. However, recent work has shown how to train the G^3 given only the correct action (Tellex et al., 2014). Additional work has modified this algorithm to search over planning constraints instead of the action space of the robot, improving inference time (Howard et al., 2014).

Dzifcak et al. (Dzifcak et al., 2009) present a CCG-based semantic parsing approach to interpreting robot commands. This system is built on top of existing action and perceptual primitives, such as GOTO. The system use an incremental, heuristic semantic parser with a small, manually built lexicon. The meaning of commands are represented both in terms of goals and actions. This approach has also been extended to handle conversational disfluencies (Cantrell et al., 2010). Skubic et al. (Skubic et al., 2004) present a similar parsing system for interpreting spatial language commands to robots.

Andreas and Klein present a model for mapping natural language text to paths in continuous spaces (Andreas and Klein, 2014). Paths are segmented and featurized, so text is grounded to a sequence of vector values. The proposed model learns to align text segments to a sequence index, then uses linear regression to predict each vector value. This model can be used for direction following, but is also applicable to tasks such as predicting the shape of a stock chart given a description of the stock's performance.

Chapter 4

Training Semantic Parsers with Distant Supervision

Abstract

This chapter presents an approach to training a semantic parser using only unlabeled sentences and a knowledge base, without any annotated logical forms or denotations. The key observation is that two sources of distant supervision can be combined to train an accurate semantic parser: *semantic supervision* from a knowledge base and *syntactic supervision* from dependency parsed sentences. An evaluation demonstrates that a semantic parser trained using this method outperforms a state-of-the-art relation extraction system and is capable of predicting the correct logical form for 56% of natural language questions against the NELL knowledge base.

The second part of this chapter describes an extension to this algorithm that trains a joint syntactic and semantic parser. This parser produces a full syntactic parse of any English sentence while simultaneously producing logical forms for phrases that have a semantic analysis. The training procedure for this parser combines semantic supervision with syntactic supervision from CCGbank. An evaluation demonstrates that this syntactic and semantic parser produces more complete semantic analyses than the previous semantic parser, and that its syntactic analyses have near state-of-the-art accuracy.

4.1 Introduction

Training semantic parsers for large knowledge bases such as NELL (Carlson et al., 2010) and Freebase (Bollacker et al., 2008) is difficult because it is laborious to annotate a sufficient quantity of training data. Furthermore, the amount of data that must be annotated grows roughly proportionally to the number of predicates in the knowledge base, because the parser must learn the various ways of referring to each one. Therefore, as knowledge bases continue to grow in the future, data annotation will only become more difficult. More automated training methods are required to produce semantic parsers for these large knowledge bases.

The first part of this chapter presents an algorithm for training a semantic parser without per-sentence annotations of either logical forms or denotations. Instead, the approach exploits two easily-obtainable sources of supervision: a large knowledge base and (automatically) dependency parsed sentences from a large corpus that includes mentions of some facts from the knowledge base. The semantic parser is trained to extract relation instances in the knowledge base from the sentences, while simultaneously producing parses that syntactically agree with their dependency parses. The minimal supervision requirements of this approach enable it to train a semantic parser for any knowledge base without annotated training data. An evaluation demonstrates that a semantic parser trained using this approach extracts binary relations with state-of-the-art performance. It also maps natural language questions to Freebase queries with 56% accuracy, despite never seeing a labeled logical form or denotation.

A natural question arises from this initial work: how do we best incorporate syntactic information into semantic parsers? Most semantic parsers today are trained for question answering domains where the short question length reduces the importance of syntactic information. However, syntactic information seems helpful for parsing more complex sentences, e.g., from newswire or the web. Indeed, the evaluation of the distantly-supervised parser demonstrates that syntactic information is necessary to produce accurate semantic parses of web sentences (Section 4.2.3). However, the proposed method for incorporating syntactic information is somewhat unsatisfactory. It uses a pipelined approach: a fixed dependency parse is given as input, and the semantic parse is constrained to follow the same syntactic structure. This method prevents any interaction between syntactic and semantic parsing, even in cases where semantic information could be used to disambiguate the syntactic structure. A possible improvement is to perform a joint syntactic and semantic parse of the whole sentence, enabling syntax and semantics to interact, and, ideally, improving both syntactic and semantic parsing performance.

The second part of this chapter presents an extension to the first algorithm that trains a joint syntactic and semantic parser. This parser produces a full syntactic analysis of any English sentence while simultaneously producing logical forms tied to a knowledge base for portions of the sentence. The training procedure uses the same distant supervision method as above, except that it replaces the dependency parsed sentences with annotated CCG syntactic trees from CCGbank (Hockenmaier and Steedman, 2002a). Incorporating the wide-coverage syntactic information from CCGbank enables the parser to parse arbitrary English sentences. Using this algorithm, I train a parser called ASP that produces logical forms tied to the NELL knowledge base. An evaluation demonstrates that ASP's syntactic parsing ability is near state-of-the-art and that it produces more complete logical forms than the first approach.

The material in this chapter is taken from (Krishnamurthy and Mitchell, 2012, 2014).

4.2 Training Semantic Parsers with Distant Supervision

This section describes a process for producing a semantic parser given a corpus of dependency-parsed sentences and a knowledge base. The inputs to this process are:

1. A knowledge base $K = (E, R, C, \Delta)$, containing entities $e \in E$, categories $c \in C$, relations $r \in R$ and predicate instances $\{c(e), r(e_1, e_2)\} \in \Delta$.
2. A collection of dependency-parsed sentences $s \in S$.

3. A procedure for identifying mentions of entities from K in sentences from S (e.g., string matching).

Given these inputs, the semantic parser is produced in two steps. The first step is lexicon and grammar induction, which is performed using a rule-based approach. This process is described in Section 4.2.1, which also describes the features used by the semantic parser. The output of this step is a parametric semantic parsing model $\Gamma(\ell, t, s; \theta)$ that defines a (non-probabilistic) score for a semantic parse (ℓ, t) of a sentence s given parameters θ. The second step estimates the parameters θ of this semantic parsing model using the knowledge base as distant supervision. Section 4.2.2 describes this training procedure. Finally, Section 4.2.3 describes experiments using this algorithm to train a semantic parser for 77 relations present in both NELL and Freebase.

4.2.1 Semantic Parsing Model

This section describes the Combinatory Categorial Grammar (CCG) semantic parser used in distantly-supervised training.

Lexicon and Grammar

The parser has a lexicon containing entries of the following forms:

$$\begin{aligned}
\text{town} \ &:= \ N : \lambda x.\textsc{city}(x) \\
\text{California} \ &:= \ N : \lambda x.x = \textsc{California} \\
\text{in} \ &:= \ (N\backslash N)/N : \lambda f.\lambda g.\lambda x.\exists y.f(y) \land g(x) \land \textsc{LocatedIn}(x, y) \\
\text{in} \ &:= \ (PP/N) : \lambda f.f \\
\text{lives} \ &:= \ (S\backslash N)/PP : \lambda f.\lambda g.\exists x, y.f(y) \land g(x) \land \textsc{HasResidence}(x, y) \\
\text{is} \ &:= \ (S\backslash N)/N : \lambda f.\lambda g.\exists x.f(x) \land g(x)
\end{aligned}$$

The lexicon includes entries for nouns, prepositions, verbs and named entities. Nouns are mapped to categories in the knowledge base, verbs and prepositions are mapped to relations, and named entities are mapped to knowledge base entities. There are also special entries for forms of "to be" that equate its two arguments. The process for generating lexicon entries is described later in this section.

The grammar of the semantic parser uses only the forward and backward application combinators. It also includes some additional type-changing rules that instantiate a relation between adjacent nouns. These rules enable the parser to analyze noun compounds; for example, they enable the parser to produce the logical form $\lambda x.\exists y.\textsc{city}(x) \wedge \textsc{LocatedIn}(x, y) \wedge y = \textsc{California}$ for the phrase "California city." The process for generating these type changing rules is described later in this section. The parser is also allowed to skip words while parsing a sentence. Word skipping is necessary because the sentences in the web corpus contain many out-of-domain words whose semantics cannot be represented using predicates from the knowledge base. (Note that the ASP parser described in Section 4.3 produces a full syntactic parse that includes these tokens.)

Logical Forms

The logical forms generated by the parser are conjunctions of predicates from the knowledge base with existentially-quantified variables and λ expressions. The syntactic types N and PP are semantically represented as functions from entities to truth values (e.g., $\lambda x.\textsc{city}(x)$), while sentences S are statements with no λ terms, such as $\exists x, y.x = \textsc{California} \wedge \textsc{city}(y) \wedge \textsc{locatedIn}(x, y)$.

Although these logical forms are more expressive than the relation instances extracted by a relation extractor, they are less expressive than the logical forms generated by other semantic parsers (e.g., (Zettlemoyer and Collins, 2005)). Other parsers can represent superlatives, negation, and other phenomena that cannot be represented as a conjunction of predicates. Some of these phenomena, such as negation, can be easily incorporated into the current parser by manu-

Part of Speech	Dependency Parse Pattern	Lexical Category Template

Proper Noun	(name of entity e)
	Sacramento
		w := $N : \lambda x.x = e$
		Sacramento := $N : \lambda x.x = \textsc{Sacramento}$

Common Noun	$e_1 \overset{SBJ}{\Longrightarrow}$ [is, are, was, ...] $\overset{OBJ}{\Longleftarrow}$ w
	Sacramento is the capital
		w := $N : \lambda x.c(x)$
		capital := $N : \lambda x.\textsc{City}(x)$

Noun Modifier	$e_1 \overset{NMOD}{\Longleftarrow} e_2$ Sacramento, California	Type change $N : \lambda x.c(x)$ to $N\vert N : \lambda f.\lambda x.\exists y.c(x) \wedge f(y) \wedge r(x, y)$ $N : \lambda x.\textsc{City}(x)$ to $N\vert N : \lambda f.\lambda x.\exists y.\textsc{City}(x) \wedge f(y) \wedge \textsc{LocatedIn}(x, y)$

Preposition	$e_1 \overset{NMOD}{\Longleftarrow}$ w $\overset{PMOD}{\Longleftarrow} e_2$
	Sacramento in California
	$e_1 \overset{SBJ}{\Longrightarrow}$ VB* $\overset{ADV}{\Longleftarrow}$ w $\overset{PMOD}{\Longleftarrow} e_2$
	Sacramento is located in California
		w := $(N\backslash N)/N : \lambda f.\lambda g.\lambda x.\exists y.f(y) \wedge g(x) \wedge r(x, y)$
		in := $(N\backslash N)/N : \lambda f.\lambda g.\lambda x.\exists y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x, y)$
		w := $PP/N : \lambda f.\lambda x.f(x)$
		in := $PP/N : \lambda f.\lambda x.f(x)$

Verb	$e_1 \overset{SBJ}{\Longrightarrow}$ w* $\overset{OBJ}{\Longleftarrow} e_2$ Sacramento governs California $e_1 \overset{SBJ}{\Longrightarrow}$ w* $\overset{ADV}{\Longleftarrow}$ [IN,TO] $\overset{PMOD}{\Longleftarrow} e_2$ Sacramento is located in California $e_1 \overset{NMOD}{\Longleftarrow}$ w* $\overset{ADV}{\Longleftarrow}$ [IN,TO] $\overset{PMOD}{\Longleftarrow} e_2$ Sacramento located in California	w* := $(S\backslash N)/N : \lambda f.\lambda g.\exists x, y.f(y) \wedge g(x) \wedge r(x, y)$ governs := $(S\backslash N)/N : \lambda f.\lambda g.\exists x, y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x, y)$ w* := $(S\backslash N)/PP : \lambda f.\lambda g.\exists x, y.f(y) \wedge g(x) \wedge r(x, y)$ is located := $(S\backslash N)/PP : \lambda f.\lambda g.\exists x, y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x, y)$ w* := $(N\backslash N)/PP : \lambda f.\lambda g.\lambda y.f(y) \wedge g(x) \wedge r(x, y)$ located := $(N\backslash N)/PP : \lambda f.\lambda g.\lambda y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x, y)$
Forms of "to be"	(none)	w* := $(S\backslash N)/N : \lambda f.\lambda g.\exists x.g(x) \wedge f(x)$

Table 4.1: Dependency parse patterns used to instantiate lexicon entries for the semantic parser lexicon Λ. Each pattern is followed by an example phrase that instantiates it. An * indicates a position that may be filled by multiple consecutive words in the sentence. e_1 and e_2 are the entities identified in the sentence, r represents a relation where $r(e_1, e_2)$, and c represents a category where $c(e_1)$. Each template may be instantiated with multiple values for the variables e, c, r.

ally defining lexicon entries for a number of key words, such as "not." Others, such as superlatives (e.g., "biggest"), are more challenging because the knowledge base does not have ordinal values for implementing these kinds of comparisons.

Lexicon and Grammar Generation

The lexicon Λ and the type-changing rules used by the semantic parser are generated by applying dependency-parse-based heuristics to the sentences in the training corpus. The first step in lexicon construction is to use the mention identification procedure to identify all mentions of entities in the sentences S. This process results in (e_1, e_2, s) triples, consisting of sentences with two entity mentions. The dependency path between e_1 and e_2 in s is then matched against the dependency parse patterns in Table 4.1 to instantiate lexicon entries and add them to Λ. Each pattern consists of a dependency path with optional part-of-speech or word restrictions for each traversed node. If the pattern matches a path in the parse, the lexicon entry template on the right is instantiated and added to the lexicon. These templates are lexicon entries containing parameters c and r that are chosen at initialization time. Given the triple (e_1, e_2, s), relations r are chosen such that $r(e_1, e_2) \in \Delta$, and categories c are chosen such that $c(e_1) \in \Delta$ or $c(e_2) \in \Delta$. The template is instantiated with every combination of these c and r values.

After creating lexicon entries from each sentence in S, infrequent entries are pruned to improve parser efficiency. This pruning step is required because the common noun pattern generates a large number of lexicon entries, the majority of which will never be used in a correct parse. Pruning eliminates all common noun categories instantiated by fewer than 5 sentences. The other rules are less fertile and therefore their output need not be pruned.

The type-changing rules included in the grammar are also determined by matching dependency parse patterns to the training data (see Table 4.1). These rules change a category of the form $N : \lambda x.f(x)$ to $N|N : \lambda g.\lambda x.\exists y.f(x) \wedge g(y) \wedge r(x, y)$. The purpose of these rules is to model the semantics of noun compounds, such as "Texas city," by instantiating a relation between the two constituent nouns.

Semantic Parser Parameterization

The semantic parser Γ is a discriminative linear model:

$$\Gamma(\ell, t, s; \theta) \quad = \quad \theta^T \phi(\ell, t, s)$$

The feature function ϕ contains two sets of features. The first set consists of lexicon features that count the number of times each lexicon entry is used in the parse. The second set consists of combinator features that count the number of times each combinator is applied to each possible set of arguments. An argument is defined by its syntactic and semantic category, and in some cases by its head word. The arguments for prepositional phrases PP and common nouns include their head word, which allows the parser to distinguish between prepositional phrases headed by different prepositions, as well as between different common nouns. All other arguments are distinguished solely by syntactic and semantic category.

4.2.2 Distantly-Supervised Training

The object of distantly-supervised training is to estimate parameters θ for a given semantic parsing model $\Gamma(\ell, t, s; \theta)$ using a collection of dependency parsed sentences S and a knowledge base K. This problem is ill-posed without additional assumptions: since the correct logical form for a sentence is never observed, there is no *a priori* reason to prefer one semantic parse to another, and therefore no reason to prefer any particular set of parameters. The training algorithm makes two assumptions about correct semantic parses, which are encoded as distant supervision constraints. These constraints make learning possible by adding an inductive bias:

1. **Semantic constraint:** Every relation instance $r(e_1, e_2) \in \Delta$ is expressed by at least one sentence in S that mentions both e_1 and e_2 (Hoffmann et al., 2011; Riedel et al., 2010).
2. **Syntactic constraint:** The correct semantic parse of a sentence s contains a subset of the syntactic dependencies in a dependency parse of s.

Distantly-supervised training uses these constraints as a proxy for labeled logical forms. The training algorithm has two steps. First, the algorithm constructs a graphical model that contains both the semantic parser and constant factors encoding the above two constraints. Next, this graphical model is used to optimize the semantic parser parameters θ to produce parses that satisfy the distant supervision constraints. If the two above assumptions are correct and sufficiently constrain the parameter space, then this procedure will identify parameters for an accurate semantic parser.

Encoding the Distant Supervision Constraints

The first step of training constructs a graphical model containing the semantic parser and the two distant supervision constraints. Let $S_{(e_1,e_2)} \subseteq S$ be the subset of sentences that mention both e_1 and e_2. The semantic constraint couples the logical forms for all sentences $S_{(e_1,e_2)}$, so the graphical model is instantiated once per (e_1, e_2) tuple. Figure 4.1 shows the graphical model

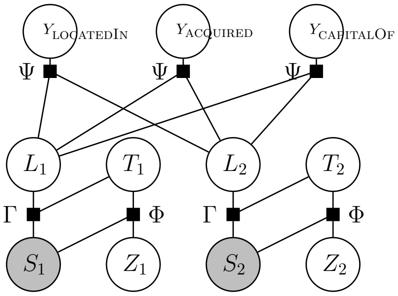


Figure 4.1: Graphical model containing the semantic parser Γ and distant supervision constraints Ψ and Φ, instantiated for an (e_1, e_2) tuple occurring in 2 sentences S_1 and S_2, with corresponding logical forms L_1 and L_2. T_1 and T_2 represent the syntactic parses of each sentence and Z_1 and Z_2 are boolean variables representing the satisfaction of the syntactic constraint Φ. The knowledge base contains 3 relations, represented by the Y variables.

instantiated in the case where $S_{(e_1,e_2)}$ contains 2 sentences and the knowledge base contains 3 relation types. The model has 5 types of random variables and values: $S_i = s_i$ represents a sentence, $T_i = t_i$ represents a syntactic CCG parse, $L_i = \ell_i$ represents a logical form, $Z_i = z_i$ represents the satisfaction of the syntactic constraint and $Y_r = y_r$ represents the truth value of relation r. S_i, T_i, L_i and Z_i are replicated once for each sentence $s \in S_{(e_1,e_2)}$, while Y_r is replicated once for each relation type r in the knowledge base. In this section, boldface variables indicate a collection of variables, e.g., $\boldsymbol{\ell} = \{\ell_1, \ell_2, ...\}$.

The graphical model is a linear model that defines a score for **L**, **T**, **Y**, **Z** given a collection of sentences **S**. This score factorizes as:

$$f(\mathbf{Y}=\mathbf{y}, \mathbf{Z}=\mathbf{z}, \mathbf{L}=\boldsymbol{\ell}, \mathbf{T}=\mathbf{t}, \mathbf{S}=\mathbf{s}; \theta) \quad = \quad \sum_r \Psi(y_r, \boldsymbol{\ell}) + \sum_i \Phi(z_i, t_i, s_i) + \Gamma(\ell_i, t_i, s_i; \theta)$$

This factorization contains three replicated factors. Γ represents the semantic parser, which is parametrized by θ and produces a semantic parse (ℓ_i, t_i) for each sentence s_i. Ψ and Φ are deterministic factors representing the two distant supervision constraints. The following sections describe each factor in more detail.

Semantic Parser

The factor Γ represents the semantic parsing model provided as part of the input (Section 4.2.1). Given a sentence s and parameters θ, the parser defines a score for a logical form ℓ and syntactic tree t using features $\phi(\ell, t, s)$:

$$\Gamma(\ell, t, s; \theta) \quad = \quad \theta^T \phi(\ell, t, s)$$

Semantic Constraint

The semantic constraint states that, given an entity tuple (e_1, e_2), every relation instance $r(e_1, e_2) \in \Delta$ must be expressed by a sentence in $S_{(e_1,e_2)}$. Furthermore, no logical form can express a relation instance which is not in Δ. This constraint is identical to the multiple deterministic-OR constraint used by Hoffmann et al. (Hoffmann et al., 2011) to train a sentential relation extractor.

The graphical model contains a semantic constraint factor Ψ and one binary variable Y_r for each relation r in the knowledge base. Y_r represents whether $r(e_1, e_2)$ is entailed by the logical form of any sentence in $S_{(e_1,e_2)}$. The Ψ factor determines whether each logical form in $\boldsymbol{\ell}$ entails the relation r between e_1 and e_2. It then aggregates these sentence-level entailments using a deterministic OR: if any sentence entails $r(e_1, e_2)$ then $Y_r = 1$. Otherwise, $Y_r = 0$. This constraint is implemented using the following formula:

$$\Psi(Y_r, \boldsymbol{\ell}) = \begin{cases} 0 & \text{if } \; Y_r = 1 \wedge \exists i.\textsc{Entails}(\ell_i, r, e_1, e_2) \\ 0 & \text{if } \; Y_r = 0 \wedge \; \nexists i.\textsc{Entails}(\ell_i, r, e_1, e_2) \\ -\infty & \text{otherwise} \end{cases}$$

$\textsc{Entails}(\ell, r, e_1, e_2)$ is true if ℓ entails the relation $r(e_1, e_2)$ and false otherwise. Applying this function to the output of a semantic parser converts it into a sentential relation extractor. Since the parser's logical forms consist of conjunctions of predicates from the knowledge base, $\textsc{Entails}(\ell, r, e_1, e_2)$ is true if ℓ contains the clauses $r(x, y)$, $x = e_1$ and $y = e_2$.

Syntactic Constraint

If used in isolation, the semantic constraint admits a large number of ungrammatical parses. The syntactic constraint rules out ungrammatical parses by forcing the semantic parser to produce syntactic parses with the same predicate-argument structure as a dependency parse of the sentence. The syntactic constraint is encoded in the graphical model by the Φ factors and **Z** variables:

$$\Phi(z, t, s) = \begin{cases} 0 & \text{if } z = \textsc{Agree}(t, \textsc{DepParse}(s)) \\ -\infty & \text{otherwise} \end{cases}$$

The AGREE function is defined as a function of the CCG dependency structures in the syntactic CCG parse t. A dependency structure is created at each rule application between the head words of the two combined subtrees; this structure is consistent with the dependency parse if the two words have a dependency edge between them (in either direction). AGREE(t, DEPPARSE(s)) is true if and only if every dependency structure in t is consistent with DEPPARSE(s).

Parameter Estimation

The training data for the graphical model contains a single training example for every tuple of entities (e_1, e_2). The input to the model is $\mathbf{s} = S_{(e_1,e_2)}$, the set of sentences containing e_1 and e_2. The distant supervision variables, **y** and **z**, are the predictions of the graphical model. **y** is constructed by setting $y_r = 1$ if $r(e_1, e_2) \in \Delta$ and 0 otherwise. This setting trains the semantic parser to extract every true relation instance between (e_1, e_2) from some sentence in $S_{(e_1,e_2)}$, while simultaneously avoiding incorrect instances. Finally, $\mathbf{z} = \mathbf{1}$, to force syntactic agreement between the semantic and dependency parses. The training data for the model is therefore a collection, $\{(\mathbf{s}^j, \mathbf{y}^j, \mathbf{z}^j)\}_{j=1}^n$, where j indexes entity tuples (e_1, e_2).

Training optimizes the semantic parser parameters θ to predict $\mathbf{Y} = \mathbf{y}^j, \mathbf{Z} = \mathbf{z}^j$ given $\mathbf{S} = \mathbf{s}^j$. The parameters θ are estimated by running the structured perceptron algorithm (Collins, 2002) on the training data defined above. The structured perceptron algorithm iteratively applies a simple update rule for each example $(\mathbf{s}^j, \mathbf{y}^j, \mathbf{z}^j)$ in the training data:

$$\begin{aligned}
\boldsymbol{\ell}^{predicted}, \mathbf{t}^{predicted} &\leftarrow \arg\max_{\boldsymbol{\ell},\mathbf{t}} \left(\max_{\mathbf{y},\mathbf{z}} f(\mathbf{y}, \mathbf{z}, \boldsymbol{\ell}, \mathbf{t}, \mathbf{s}^j; \theta^t) \right) \\
\boldsymbol{\ell}^{actual}, \mathbf{t}^{actual} &\leftarrow \arg\max_{\boldsymbol{\ell},\mathbf{t}} f(\mathbf{y}^j, \mathbf{z}^j, \boldsymbol{\ell}, \mathbf{t}, \mathbf{s}^j; \theta^t) \\
\theta^{t+1} &\leftarrow \theta^t + \sum_i \phi(\ell_i^{actual}, t_i^{actual}, s_i) - \sum_i \phi(\ell_i^{predicted}, t_i^{predicted}, s_i)
\end{aligned}$$

Each iteration of training requires solving two maximization problems. The first maximization, $\max_{\mathbf{y},\mathbf{z},\boldsymbol{\ell},\mathbf{t}} f(\mathbf{y}, \mathbf{z}, \boldsymbol{\ell}, \mathbf{t}, \mathbf{s}^j; \theta^t)$, is straightforward because **y** and **z** are deterministic functions of $\boldsymbol{\ell}$ and **t**. Therefore, this problem can be solved by finding the maximum weight semantic parses $(\boldsymbol{\ell}, \mathbf{t})$ given $\mathbf{s}^j$, then choosing values for **y** and **z** that satisfy the distant supervision constraints.

The second maximization, $\max_{\boldsymbol{\ell},\mathbf{t}} f(\mathbf{y}^j, \mathbf{z}^j, \boldsymbol{\ell}, \mathbf{t}, \mathbf{s}^j; \theta^t)$, is more challenging. When $\mathbf{y}^j$ and $\mathbf{z}^j$ are given, the inference procedure must restrict its search to the parses $(\boldsymbol{\ell}, \mathbf{t})$ that satisfy the distant supervision constraints. Although the original formulation of the Ψ factors permitted tractable inference, the inclusion of the ENTAILS function precludes efficient inference. However, this maximization can be approximated using beam search. The approximate inference algorithm first performs a beam search to produce $k = 300$ possible semantic parses for each sentence s. Next, it checks the value of Φ for each generated parse and eliminates parses which do not satisfy the syntactic constraint.[1] Finally, it applies the ENTAILS function to each parse, then uses the greedy approximate inference procedure from Hoffmann et al. (Hoffmann et al., 2011) for the Ψ factors. This procedure can occasionally fail to find a collection of semantic parses that satisfy both constraints; if this happens during training, the current training example is skipped.

[1] Note that the syntactic constraint Φ can also be enforced during the beam search as part of the CKY algorithm. This choice would reduce the number of search errors by guaranteeing that every parse in the beam satisfies the syntactic constraint. However, given the limited grammar of the semantic parser, such enforcement was not necessary.

4.2.3 Evaluation

This section evaluates the performance of a semantic parser for NELL trained using distant supervision. Empirical comparison is somewhat difficult because the most comparable previous work – distantly-supervised relation extraction – uses a shallower semantic representation. The evaluation therefore has two components: (1) a binary relation extraction task, to demonstrate that the trained semantic parser extracts instances of binary relations with performance comparable to other state-of-the-art systems, and (2) a natural language database query task, to demonstrate the parser's ability to extract more complex logical forms than binary relation instances, such as conjunctions of multiple categories and relations with partially shared arguments.

Data

The experiments use a subset of 77 relations from NELL that are also present in Freebase.[2] All entities and predicate instances were taken from Freebase to eliminate the effect of NELL's potentially noisy knowledge base. The corpus of sentences was randomly sampled from a web crawl[3] and dependency parsed with MaltParser (Nivre et al., 2006). Long sentences tended to have noisy parses while also rarely expressing relations, so sentences longer than 10 words were discarded. Entities were identified by performing a simple string match between canonical entity names in Freebase and proper noun phrases identified by the parser. In cases where a single noun phrase matched multiple entities, the matching algorithm chose the entity participating in the most relations. The resulting corpus contains 2.5 million (e_1, e_2, s) triples, with 10% reserved for validation and 10% for testing. The validation set was used to estimate performance during algorithm development, while the test set was used to generate the final experimental results. All triples for each (e_1, e_2) tuple were placed in the same set.

Approximately 1% of the resulting (e_1, e_2, s) triples are positive examples, meaning there

[2] These relations are defined by a set of MQL queries and potentially traverse multiple Freebase relation links.

[3] The web crawl is a preliminary version of the ClueWeb 2009 corpus.

Relation Name	Relation Instances	Sentences
cityLocatedInState	2951	13422
cityLocatedInCountry	1696	7904
cityOfPersonBirth	397	440
companiesHeadquarteredHere	326	432
musicArtistMusician	251	291
cityUniversities	239	338
cityCapitalOfCountry	123	2529
hasHusband	103	367
parentOfPerson	85	356
hasSpouse	81	461

Table 4.2: Occurrence statistics for the 10 most frequent relations in the training data. Relation Instances shows the number of entity tuples (e_1, e_2) that appear as positive examples for each relation, and Sentences shows the total number of sentences in which these tuples appear.

exists some relation r where $r(e_1, e_2) \in \Delta$. To improve training efficiency and prediction performance, only 5% of the negative examples were used for training, producing a training set of 125k sentences with 27k positive examples. The validation and test sets retain the original positive/negative ratio. Table 4.2 shows some statistics of the most frequent relations in the test set.

Relation Extraction Evaluation

The first experiment measures the semantic parser's ability to extract relations from sentences in the web corpus. The parser is compared to MultiR(Hoffmann et al., 2011) a state-of-the-art distantly-supervised relation extractor. This extractor uses the same distant supervision constraint and parameter estimation procedure, but replaces the semantic parser with a linear classifier. The features for this classifier include the dependency path between the entity mentions, the type of each mention, and the intervening context (Mintz et al., 2009).

Both the semantic parser and MultiR were trained by running 5 iterations of the structured perceptron algorithm.[4] At test time, both models predicted a relation $r \in R$ or NONE for each

[4] The structured perceptron algorithm does not converge to a parameter estimate and performance did not seem to improve beyond 5 iterations. The final iterate of the algorithm was used as the parameter estimate.

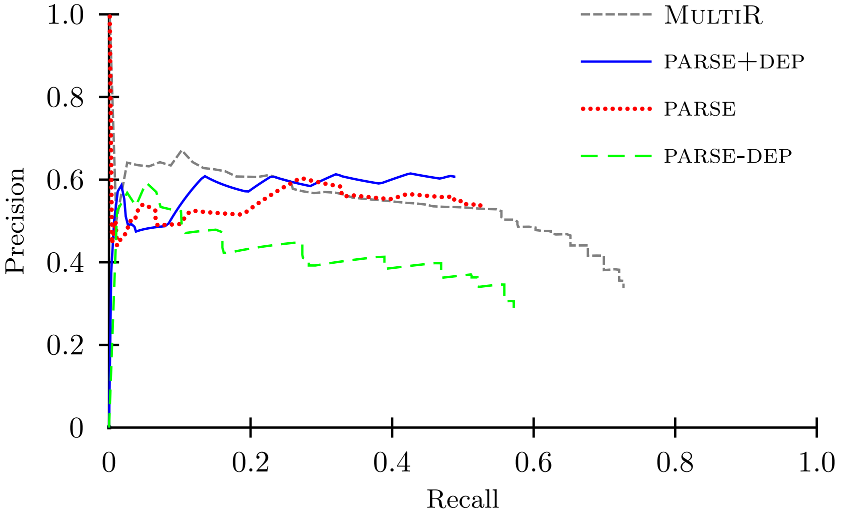


Figure 4.2: Aggregate precision as a function of recall for MULTIR and the three semantic parser variants.

(e_1, e_2, s) triple in the test set. To make these predictions, the parser parses the sentence without any knowledge of the particular entities under consideration, then applies the ENTAILS function defined in Section 4.2.2 to determine which relation (if any) holds between e_1 and e_2. The experiments compare three versions of the semantic parser: PARSE, which is the basic semantic parser, PARSE+DEP which additionally observes a dependency parse at test time, and PARSE-DEP which is trained without the syntactic constraint. Note that MULTIR is most comparable to PARSE+DEP, as MULTIR also uses a dependency parse of the sentence to construct its feature vector.

The evaluation considers two performance measures: aggregate and sentential precision/recall. Aggregate precision takes the union of all extracted relation instances $r(e_1, e_2)$ from the test corpus and compares these instances to the knowledge base. To produce a precision/recall curve, each extracted instance $r(e_1, e_2)$ is assigned the maximum score over all sentences which extracted it. This metric is easy to compute, but is a pessimistic performance estimate due to missing relation instances in the knowledge base.

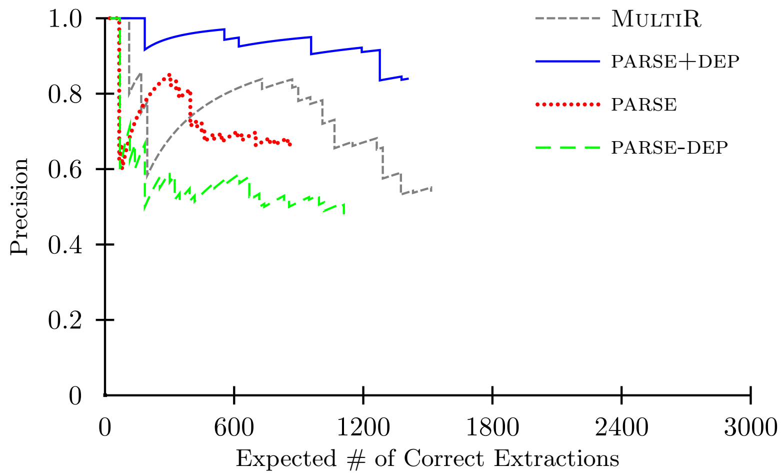


Figure 4.3: Sentential precision as a function of the expected number of correct extractions for MULTIR and the three semantic parser variants. The expected number of correct extractions is directly proportional to recall.

Sentential precision measures the precision of the predicted relations for each (e_1, e_2, s) tuple. This metric is evaluated by manually sampling and evaluating 100 test sentences per model from which a relation was extracted. Unfortunately, it is difficult to compute recall for this metric, since the true number of sentences expressing relations is unknown. Therefore, the experiments report precision as a function of the expected number of correct extractions, which is directly proportional to recall. The expected number of correct extractions at rank k in the list is k multiplied by the precision at rank k.

Figure 4.2 displays aggregate precision/recall and Figure 4.3 displays sentential precision/recall for all 4 models. Generally, PARSE behaves like MULTIR with somewhat lower recall. In the sentential evaluation, PARSE+DEP outperforms both PARSE and MULTIR. The difference between PARSE+DEP's aggregate and sentential precision stems from the fact that PARSE+DEP extracts each relation instance from more sentences than either MULTIR or PARSE. The difference in performance between PARSE+DEP, PARSE and PARSE-DEP clearly demonstrates the value of syntactic information, which improves performance when incorporated both at training and test time. Precision in the aggregate experiment is low partially due to examples with incorrect en-

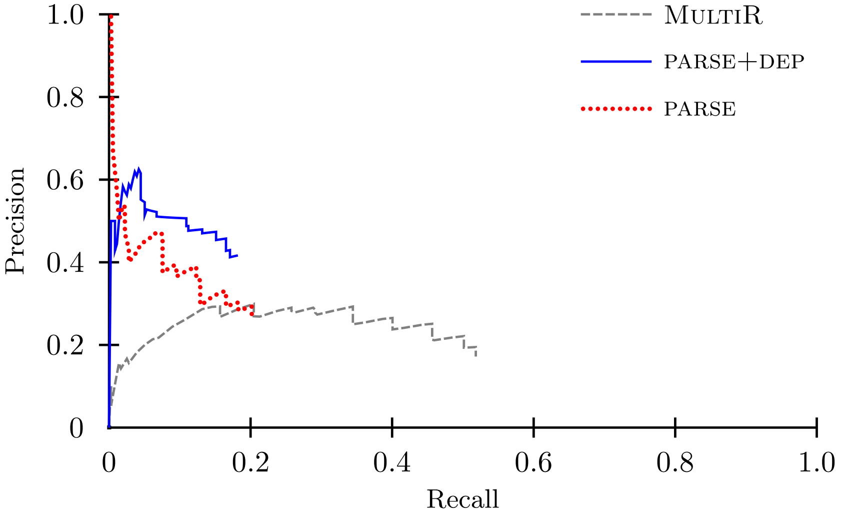


Figure 4.4: Aggregate precision as a function of recall, ignoring the two most frequent relations, cityLocatedInState and cityLocatedInCountry.

tity disambiguation. Many Freebase entities have names that occur frequently in text but *not* as mentions of these entities. The parser often predicts that these occurrences are entity mentions, leading to incorrect extractions.

The skewed distribution of relation types also hides interesting differences between the models. Figure 4.4 compares the two syntactically-supervised parsers to MultiR, ignoring the two most frequent relations (which together make up over half of all relation instances). Both parse and parse+dep are considerably more precise than MultiR on these less frequent relations because the semantic parsers can share parameters between relations due to the compositional structure of their parses. For example, the semantic parsers learn that "in" often combines with a city to form a prepositional phrase; the parsers can apply this knowledge to identify city arguments of any relation. However, MultiR is capable of higher recall, since its dependency parse features can represent syntactic dependencies that cannot be represented by the semantic parsers. This limitation of the semantic parsers is a consequence of the heuristic lexicon initialization procedure and could be rectified by a more sophisticated procedure.

Example Query	Logical Form
capital of Russia	$\lambda x.\textsc{cityCapitalOfCountry}(x, \textsc{Russia})$
wife of Abraham	$\lambda x.\textsc{hasHusband}(x, \textsc{Abraham})$
vocalist from London, England	$\lambda x.\textsc{musician}(x)\wedge$ $\textsc{personBornIn}(x, \textsc{London})\wedge$ $\textsc{cityInCountry}(\textsc{London}, \textsc{England})$
home of ConocoPhillips in Canada	$\lambda x.\textsc{headquarters}(\textsc{ConocoPhillips}, x)$ $\wedge\textsc{cityInCountry}(x, \textsc{Canada})$

Table 4.3: Example natural language questions and their annotated logical forms.

Semantic Parsing Evaluation

The second experiment measures the trained parser's ability to correctly translate natural language questions into logical queries against the knowledge base.

To avoid biasing the evaluation, the test corpus of questions was constructed in a data-driven fashion. The test data was searched for sentences with two related entities separated by an "is a" expression. The portion of the sentence before the "is a" expression was discarded and the remainder retained as a candidate question. For example "Jesse is an author from Austin, Texas," was converted into the candidate question "author from Austin, Texas." Each candidate question was then annotated with a logical form using categories and relations from the knowledge base; candidate questions without satisfactory logical forms were discarded. 50 validation and 50 test questions were annotated in this fashion. The validation set was used to estimate performance during algorithm development and the test set was used to generate the final results. Example questions with their annotated logical forms are shown in Table 4.3.

Table 4.4 displays the results of the semantic parsing evaluation. Precision is the percentage of successfully parsed questions for which the correct logical form was predicted. Recall is the percentage of all questions for which the correct logical form was predicted. This evaluation demonstrates that the semantic parser successfully interprets common nouns and identifies mul-

	Precision	Recall
PARSE	0.80	0.56
PARSE-DEP	0.45	0.32

Table 4.4: Precision and recall for predicting logical forms of natural language queries against Freebase. The table compares PARSE, trained with syntactic supervision to PARSE-DEP, trained without syntactic supervision.

tiple relations with shared arguments. The performance difference between PARSE and PARSE-DEP also demonstrates the benefit of including syntactic supervision. PARSE+DEP was excluded from this evaluation as the construction of the test questions did not preserve the dependency parse structure of the original sentence. For this evaluation, both parsers were prevented from skipping words.

Examining the system output, there are two major sources of error. The first is missing lexicon entries for uncommon words (e.g., "ex-guitarist"), which negatively impact recall by making some queries unparsable. The second is difficulty distinguishing between relations with similar type signatures, such as CITYLOCATEDINCOUNTRY and CITYCAPITALOFCOUNTRY.

4.3 Joint Syntactic and Semantic Parsing

This section describes an algorithm for producing a joint syntactic and semantic parser. This parser produces a full syntactic parse of every sentence and also produces logical forms for portions of the sentence that have a semantic representation within the parser's predicate vocabulary. The inputs to this algorithm are:

1. A knowledge base $K = (E, R, C, \Delta)$, containing entities $e \in E$, categories $c \in C$, relations $r \in R$ and predicate instances $\{c(e), r(e_1, e_2)\} \in \Delta$.
2. A collection of sentences S.
3. A collection $L = \{(s_i, t_i)\}_{i=1}^{n}$ of sentences s_i with annotated syntactic CCG parse trees t_i (e.g., CCGbank).

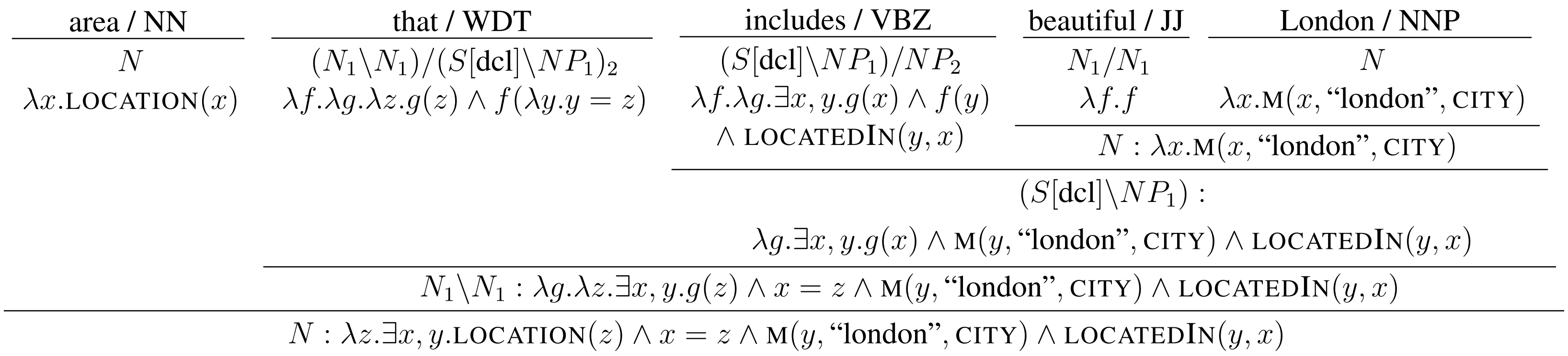
4. A procedure for identifying mentions of entities from K in sentences from S (e.g., string matching).

As in the previous section, there are two steps in producing the parser. The first step generates a semantic parsing model $\Gamma(\ell, d, t, s; \theta)$ using grammatical information from CCGbank and lexicon induction heuristics. This semantic parsing model is considerably more sophisticated than the model from the previous section due to the added challenge of producing full syntactic parses. The second training step uses the same distant supervision constraint from Section 4.2.2 to teach the parser semantics, but includes an additional term encouraging the parser to correctly parse CCGbank sentences.

I demonstrate this approach by training a joint syntactic and semantic parser named ASP. ASP produces a full syntactic analysis of every sentence while simultaneously producing logical forms containing any of 61 category and 69 relation predicates from NELL. Experiments with ASP demonstrate that jointly analyzing syntax and semantics improves semantic parsing performance over prior work (Section 4.2) and a pipelined syntax-then-semantics approach. However, the experiments also demonstrate that incorporating semantic information slightly reduces syntactic parsing performance (by $\sim 0.5\%$). Despite this drop, ASP's syntactic parsing performance is within 2.5% of state-of-the-art.

4.3.1 The ASP Parsing Model

This section describes the Combinatory Categorial Grammar (CCG) parsing model used by ASP. The input to the parser is a part-of-speech tagged sentence and the output is a syntactic CCG parse tree, along with zero or more logical forms representing the semantics of subspans of the sentence. The parser also outputs a collection of dependency structures summarizing the sentence's predicate-argument structure. Figure 4.5 illustrates ASP's input/output specification.


area / NN
N
$\lambda x.\textsc{location}(x)$
that / WDT
$(N_1\backslash N_1)/(S[\text{dcl}]\backslash NP_1)_2$
$\lambda f.\lambda g.\lambda z.g(z) \wedge f(\lambda y.y = z)$
includes / VBZ
$(S[\text{dcl}]\backslash NP_1)/NP_2$
$\lambda f.\lambda g.\exists x, y.g(x) \wedge f(y)$
$\wedge\ \textsc{locatedIn}(y, x)$
beautiful / JJ
N_1/N_1
$\lambda f.f$
London / NNP
N
$\lambda x.\textsc{m}(x, \text{“london”}, \textsc{city})$
$N : \lambda x.\textsc{m}(x, \text{“london”}, \textsc{city})$
$(S[\text{dcl}]\backslash NP_1) :$
$\lambda g.\exists x, y.g(x) \wedge \textsc{m}(y, \text{“london”}, \textsc{city}) \wedge \textsc{locatedIn}(y, x)$
$N_1\backslash N_1 : \lambda g.\lambda z.\exists x, y.g(z) \wedge x = z \wedge \textsc{m}(y, \text{“london”}, \textsc{city}) \wedge \textsc{locatedIn}(y, x)$
$N : \lambda z.\exists x, y.\textsc{location}(z) \wedge x = z \wedge \textsc{m}(y, \text{“london”}, \textsc{city}) \wedge \textsc{locatedIn}(y, x)$

Head						Argument			
word	POS	semantic type	index	syntactic category	arg. num.	word	POS	semantic type	index
that	WDT	—	1	$(N_1\backslash N_1)/(S\backslash NP_1)_2$	1	area	NN	LOCATION	0
that	WDT	—	1	$(N_1\backslash N_1)/(S\backslash NP_1)_2$	2	includes	VBZ	LOCATEDIN^{-1}	2
includes	VBZ	LOCATEDIN^{-1}	2	$(S[\text{dcl}]\backslash NP_1)/NP_2$	1	area	NN	LOCATION	0
includes	VBZ	LOCATEDIN^{-1}	2	$(S[\text{dcl}]\backslash NP_1)/NP_2$	2	ENTITY:CITY	NNP	CITY	4
beautiful	JJ	—	3	N_1/N_1	1	ENTITY:CITY	NNP	CITY	4

Figure 4.5: Example input and output for ASP. Given a POS-tagged sentence, the parser produces a CCG syntactic tree and logical form (top), and a collection of dependency structures (bottom).

Lexicon

ASP uses a lexicalized and semantically-typed Combinatory Categorial Grammar (CCG) (Steedman, 1996). Example lexicon entries are:

person := N : PERSON : $\lambda x.\text{PERSON}(x)$

London := N : CITY : $\lambda x.\text{M}(x, \text{“london”}, \text{CITY})$

great := N_1/N_1 : — : $\lambda f.\lambda x.f(x)$

bought := $(S[\text{dcl}]\backslash NP_1)/NP_2$: ACQUIRED : $\lambda f.\lambda g.\exists x, y.f(y) \wedge g(x) \wedge \text{ACQUIRED}(x, y)$

Each lexicon entry maps a word to a syntactic category, semantic type, and logical form. The semantic type is a new element of the lexicon entry (relative to the lexicon described in Section 4.2.1) whose value is a category or relation predicate. These values are used by ASP to enforce type constraints during parsing and to include semantics in the parser's parameterization. Lexicon entries can have the semantic type "—" to indicate words whose semantics cannot be expressed using predicates from the ontology. The M predicate is used to represent entity mentions; see the following section on logical forms for a full description.

The syntactic categories in ASP also use the feature and head passing mechanisms of CCG. Briefly, atomic categories may have an associated feature, shown in square brackets, e.g., $S[\text{dcl}]$. A category can also associate a variable with a feature, which enables features to be passed between categories using unification. The subscripts on each category are used to populate predicate-argument dependencies and to pass head word information using unification. For example, the head word of the parse in Figure 4.5 is "area," due to the coindexing of the argument and return categories in the category $N_1\backslash N_1$. See Section 2.2.5 for a full description of features and head passing.

Dependency Structures

Parsing a sentence produces a collection of dependency structures that summarize the predicate-argument structure of the sentence. These dependency structures are the standard CCG syntactic dependencies (Clark and Curran, 2007a; Hockenmaier and Steedman, 2002a) augmented with additional fields to capture semantic information. Dependency structures are 10-tuples, of the form:

$<$ head word, head POS, head semantic type, head word index, head word syntactic category, argument number, argument word, argument POS, argument semantic type, argument word index $>$

A dependency structure captures a relationship between a head word and its argument. The syntactic category and the argument number determine which argument to the head is being populated. Dependency structures also contain word index information that allows the parser to include the distance between words in its parameterization. During parsing, whenever a subscripted argument of a syntactic category is filled, a dependency structure is created between the head of the applied category and its argument. For example, in Figure 4.5, the first application fills argument 1 of "beautiful" with "London," creating a dependency structure.

Logical Forms

ASP performs a best-effort semantic analysis of every parsed sentence, producing logical forms for subspans of the sentence when possible. Logical forms are designed so that the meaning of a sentence is a universally- and existentially-quantified conjunction of predicates with shared arguments. This representation allows the parser to produce semantic analyses for a reasonable subset of language, including prepositions, verbs, nouns, relative clauses, and conjunctions.

Figure 4.5 shows a representative sample of a logical form produced by ASP. Generally, the parser produces a lambda calculus statement with several existentially-quantified variables ranging over entities in the knowledge base. The only exception to this rule is conjunctions, which are represented using a scoped universal quantifier over the conjoined predicates. Entity mentions appear in logical forms via a special mention predicate, M, instead of as database constants. For example, "London" appears as $\text{M}(x, \text{“london”}, \text{CITY})$, instead of as a constant like LONDON. The meaning of this mention predicate is that x is an entity which can be called "london" and belongs to the CITY category. This representation propagates uncertainty about entity references into the logical form where background knowledge can be used for disambiguation. For example, "London, England" is assigned a logical form that disambiguates "London" to a "London" located in "England."[5] This mention predicate solves the entity disambiguation problem that was a frequent source of errors in Section 4.2.3.

Lexicon entries with the semantic type "—" are automatically assigned logical forms based on their head passing markup. For example, in Figure 4.5, the adjective "beautiful" is assigned $\lambda f.f$. This approach allows a logical form to be derived for most sentences, but (somewhat counter-intuitively) can lose interesting logical forms from constituent subspans. For example, the preposition "in" has syntactic category $(N_1 \backslash N_1)/N_2$, which results in the logical form $\lambda f.\lambda g.g$. This logical form discards any information present in the argument f. A simple workaround to this problem is to extract a logical form from every subtree of the CCG parse.

Parameterization

The parser Γ is trained as a discriminative linear model of the form:

$$\Gamma(\ell, d, t, s; \theta) \quad = \quad \theta^T \phi(d, t, s)$$

Given a parameter vector θ and a sentence s, the parser produces a score for a syntactic parse tree t, a collection of dependency structures d and a logical form ℓ. The feature function ϕ contains four classes of features: lexicon features, combinator features, dependency features and dependency distance features (see Table 4.5). These features are based on those of C&C (Clark and Curran, 2007a), but modified to include semantic types. The lexicon features count occurrences of each lexicon entry, and also include backoff features based on POS tags. The combinator features count the number of times that each binary and unary combinator rule is used in a parse. There is also an indicator feature for the parse tree's root syntactic category. The dependency features count the number of times each filled dependency is instantiated by a

[5] Specifically, $\lambda x.\exists y.\textsc{citylocatedincountry}(x, y) \wedge \textsc{m}(x, \text{"london"}, \textsc{city}) \wedge \textsc{m}(y, \text{"england"}, \textsc{country})$

parse, including various backoff features to POS tags. Finally, the dependency distance features count the number of intervening words, punctuation marks, and verbs between a predicate and its argument.

Importantly, the features are designed to backoff from syntactic and semantic information to purely syntactic information. Note that every feature template in Table 4.5 that depends on semantic type (either h_t or a_t) has a corresponding backoff feature with the semantic type omitted. This backoff property enables syntactic information from CCGbank to be transferred to semantic parsing. Preliminary experiments without these backoff features led to extremely poor semantic parsing results.

The parser also includes a hard type-checking constraint to ensure that logical forms are well-typed. This constraint states that dependency structures with a head semantic type only accept arguments that (1) have a semantic type and (2) are within the domain/range of the head type. This constraint is enforced during parsing by assigning a weight of $-\infty$ to any parse that instantiates a dependency structure violating either of these two rules.

Supertagging

Parsing in practice can be slow because the parser's lexicalized grammar permits a large number of parses for a sentence. *Supertagging* is a method for improving parsing speed by pruning the set of allowed syntactic categories for each token before parsing (see Section 2.1.5). ASP's supertagger is a logistic regression classifier that predicts the syntactic category of each token from features of the surrounding tokens and POS tags. Subsequent parsing is restricted to only consider categories whose probability is within a factor of α of the highest-scoring category. The parser uses a backoff strategy, first attempting to parse with the supertags from $\alpha = 0.01$, backing off to $\alpha = 0.001$ if the initial parsing attempt fails.

The supertagger is trained on Sections 02-21 of CCGbank. An evaluation on Section 00 shows that at $\alpha = 0.01$, the correct syntactic category is within the predicted set of categories for

Lexicon features: word, POS $:= X : t : \ell$	
Word/syntactic category	word, X
POS/syntactic category	POS, X
Word semantics	word, X, t

Combinator features: $X\ Y \rightarrow Z$ or $X \rightarrow Z$	
Binary combinator indicator	$X\ Y \rightarrow Z$
Unary combinator indicator	$X \rightarrow Z$
Root syntactic category	Z

Dependency Features: $< h_w, h_p, h_t, h_i, s, n, a_w, a_p, a_t, a_i >$	
Predicate-Argument Indicator	$< h_w, —, h_t, —, s, n, a_w, —, a_t, — >$
Word-Word Indicator	$< h_w, —, —, —, s, n, a_w, —, —, — >$
Predicate-POS Indicator	$< h_w, —, h_t, —, s, n, —, a_p, —, — >$
Word-POS Indicator	$< h_w, —, —, —, s, n, —, a_p, —, — >$
POS-Argument Indicator	$< —, h_p, —, —, s, n, a_w, —, a_t, — >$
POS-Word Indicator	$< —, h_p, —, —, s, n, a_w, —, —, — >$
POS-POS Indicator	$< —, h_p, —, —, s, n, —, a_p, —, — >$

Dependency Distance Features:		
Token distance	$h_w, h_t, —, s, n, d$	$d =$ Number of tokens between h_i and a_i: 0, 1, 2 or more.
Token distance word backoff	$h_w, —, s, n, d$	$d =$ Number of tokens between h_i and a_i: 0, 1, 2 or more.
Token distance POS backoff	$—, —, h_p, s, n, d$	$d =$ Number of tokens between h_i and a_i: 0, 1, 2 or more.

(The above distance features are repeated using the number of intervening verbs and punctuation marks.)

Table 4.5: Listing of parser feature templates used in the feature function ϕ. Each feature template represents a class of indicator features that fire during parsing when lexicon entries are used, combinators are applied, or dependency structures are instantiated.

98.3% of tokens. Furthermore, the supertagger predicts an average of 2.4 syntactic categories per token. At $\alpha = 0.001$, the correct syntactic category is within the predicted set for 98.9% of tokens, and an average of 5.7 syntactic categories are predicted per token. This supertagger is not as accurate as other supertaggers that model sequential sentence structure. For reference, the supertagger of the C&C parser predicts an average of 1.72 syntactic categories per token at a 98.3% accuracy level, and 3.57 syntactic categories per token at a 99.2% accuracy level (Clark and Curran, 2007a).

Lexicon and Grammar Generation

The grammar for ASP contains the annotated lexicon entries and grammar rules in Sections 02-21 of CCGbank and additional semantic entries produced using a set of dependency parse heuristics.

The lexicon Λ contains all words that occur at least 20 times in CCGbank. Any word occurring fewer than 20 times was replaced by the symbol $\mathrm{UNK\text{-}[POS]}$, where $\mathrm{[POS]}$ is its part-of-speech tag in the sentence. The resulting lexicon contains 9,528 unique words and 33,665 lexicon entries. The head passing and dependency markup was generated using the rules of the C&C parser (Clark and Curran, 2007a). These lexicon entries were then automatically annotated with logical forms based on their head passing markup. For example, the adjective category N_1/N_1 is annotated with the logical form $\lambda f.f$. All lexicon entries derived from CCGbank are assigned the semantic type —.

This lexicon is augmented with additional entries whose logical forms contain NELL predicates. These entries are instantiated using the dependency parse patterns listed in Table 4.6. These patterns operate in the same manner as those in Section 4.2.1. When applied to the training corpus, these patterns heuristically identify verbs, prepositions, and possessives that express relations, and nouns that express categories. The patterns also include special cases for forms of “to be.” This process generates ∼4000 entries (not counting entity names), representing 69 relations and 61 categories from NELL.

The parser's combinators include function application, composition, and crossed composition, as well as several binary and unary type-changing rules that occur in CCGbank. All combinators were restricted to only apply to categories that combine in Sections 02-21; this restriction has been shown to improve parsing speed without a material effect on syntactic parsing performance (Clark and Curran, 2007a). Finally, the grammar includes a number of heuristically-instantiated binary rules of the form $, \; N \rightarrow N\backslash N$ that instantiate a relation between adjacent nouns. These rules are similar to the noun compound rules from Section 4.2.1.

4.3.2 Training ASP

The training procedure for ASP is a refinement of the method from Section 4.2.2 that replaces the syntactic constraint with a collection of annotated syntactic CCG parses L. The training procedure constructs a joint objective function $O(\theta)$ that decomposes into syntactic and semantic components: $O(\theta) = O_{\text{syn}}(\theta) + O_{\text{sem}}(\theta)$. The syntactic component O_{syn} is a standard syntactic parsing objective constructed using the syntactic resource L. The semantic component O_{sem} is a distantly-supervised relation extraction task based on the semantic constraint from Section 4.2.2. Minimizing this joint objective trains the parser to simultaneously produce accurate syntactic parses while also extracting knowledge base relations. These components of the objective function are described in more detail in the following sections.

Syntactic Objective

The syntactic objective is the structured perceptron objective instantiated for a syntactic parsing task. This objective encourages the parser to accurately reproduce the syntactic parses in the annotated corpus $L = \{(s_i, t_i)\}_{i=1}^{n}$:

$$O_{\text{syn}}(\theta) \quad = \quad \sum_{i=1}^{n} |\max_{\hat{\ell},\hat{d},\hat{t}} \Gamma(\hat{\ell}, \hat{d}, \hat{t}, s_i; \theta) - \max_{\ell^*, d^*} \Gamma(\ell^*, d^*, t_i, s_i; \theta)|_+$$

Semantic Type : Logical Form Template	Syntactic Category	Dependency Parse Pattern	Example
$c : \lambda x.\textsc{m}(x, w, c)$	N	$w =$ name for entity e	London := N : $\textsc{city}$: $\lambda x.\textsc{m}(x,$ “london”, $\textsc{city})$
$c : \lambda x.c(x)$	N	$e \Rightarrow$ [form of to be] $\Leftarrow$ w [NN] $e \xLeftarrow{appos}$ w [NN]	town := N : $\textsc{city}$: $\lambda x.\textsc{city}(x)$
$r : \lambda g.\lambda f.\exists x, y.f(x) \land g(y) \land r(x, y)$	$(S[Z]\backslash N_1)/N_2$	$e_1 \xRightarrow{nsubj}$ w $\xLeftarrow{dobj} e_2$	bought := $(S[\text{dcl}]\backslash NP_1)/NP_2$: $\textsc{acquired}$: $\lambda g.\lambda f.\exists x, y.f(x) \land g(y) \land \textsc{acquired}(x, y)$
	$(S[Z]\backslash N_1)/PP_2$	$e_1 \xLeftrightarrow{nsubj}$ w $\xLeftarrow{prep}$ [IN, TO] $\Leftarrow e_2$	established := $(S[\text{pss}]\backslash NP_1)/PP_2$: $\textsc{headquarteredIn}$: $\lambda g.\lambda f.\exists x, y.f(x) \land g(y) \land \textsc{headquarteredIn}(x, y)$
$r : \lambda g.\lambda f.\lambda x.\exists y.f(x) \land g(y) \land r(x, y)$	$(NP_1\backslash NP_1)/PP_2$, $(N_1\backslash N_1)/PP_2$	$e_1 \xLeftarrow{partmod}$ w [VBN] $\Leftarrow$ [IN, TO] $\Leftarrow e_2$	born := $(NP_1\backslash NP_1)/PP_2)$: $\textsc{personBornInLocation}$: $\lambda g.\lambda f.\lambda x.\exists y.f(x) \land g(y) \land \textsc{personBornInLocation}(x, y)$
$c : \lambda f.f$	PP/NP_1	(prepositions in the above patterns)	in := PP/NP_1 : $\textsc{location}$: $\lambda f.f$
$r : \lambda g.\lambda f.\lambda x.\exists y.f(x) \land g(y) \land r(x, y)$	$(NP_1\backslash NP_1)/NP_2$, $(N_1\backslash N_1)/N_2$	$e_1 \Leftarrow$ w [IN, TO] $\Leftarrow e_2$ $e_1 \Leftrightarrow$ [be] $\Leftrightarrow$ [NN] $\Leftrightarrow$ w [IN, TO] $\Leftrightarrow e_2$ $e_1 \xLeftarrow{appos}$ [NN] $\Leftrightarrow$ w [IN, TO] $\Leftrightarrow e_2$	in := $(N_1\backslash N_1)/N_2$: $\textsc{musicianInMusicArtist}$: $\lambda g.\lambda f.\lambda x.\exists y.f(x) \land g(y) \land \textsc{musicianInMusicArtist}(x, y)$
$r : \lambda g.\lambda f.\lambda x.\exists y.f(x) \land g(y) \land r(x, y)$	$(NP_1/N_1)\backslash NP_2$	$e_1 \xLeftarrow{poss}$ w $\Rightarrow e_2$ $e_1 \Rightarrow$ [be] $\Leftarrow$ [NN] $\xLeftarrow{poss}$ w $\Rightarrow e_2$	’s := $(NP_1/N_1)\backslash NP_2$: $\textsc{directorDirectedMovie}^{-1}$: $\lambda g.\lambda f.\lambda x.\exists y.f(x) \land g(y) \land \textsc{directorDirectedMovie}(y, x)$
$\textsc{equals} : \lambda g.\lambda f.\exists x, y.f(x) \land g(y) \land x = y$	$(S[\text{dcl}]\backslash N_1)/N_2$	[forms of “to be”]	is := $(S[\text{dcl}]\backslash N_1)/N_2$: $\textsc{equals}$: $\lambda g.\lambda f.\exists x, y.f(x) \land g(y) \land x = y$
Create binary rule , $N \rightarrow N\backslash N$, where the returned logical form is: $\lambda f.\lambda x.\exists y.f(x) \land g(y) \land r(x, y)$ (where g is the logical form for the initial N)		$e_1 \xLeftarrow{appos} e_2$	, $N \rightarrow N\backslash N$: $\lambda f.\lambda x.\exists y.f(x) \land g(y) \land \textsc{cityLocatedInState}(x, y)$ (e.g., to parse “Sacramento, California”)

Table 4.6: Dependency parse heuristics for CCG lexicon induction. Each heuristic is applied to a sentence containing mentions of a pair of entities e_1, e_2. The logical form templates on the left contain categories c and relations r that are filled using predicates from the knowledge base such that $c(e)$ and $r(e_1, e_2)$. Dependency parse patterns are specified using directed parse tree edges, where bidirectional edges match either edge orientation. Bracketed expressions contain part-of-speech tag restrictions for the matched token. Example lexicon entries produced by applying the patterns are shown on the right. Noun lexicon entries generated using the second rule must be generated at least twice in the corpus to be included in the lexicon.

The first term in the above expression represents the best CCG parse of the sentence s_i according to the current model. The second term is the best parse of s_i whose syntactic tree equals the annotated syntactic tree t_i. In the above equation $|\cdot|_+$ denotes the positive part of the expression. Minimizing this objective therefore finds parameters θ that reproduce the annotated syntactic trees. Variables with a hat (e.g., $\hat{\ell}$) denote unconstrained model predictions and variables with a star (e.g., ℓ^*) denote constrained model predictions given the correct syntactic parse tree t_i.

Semantic Objective

The semantic objective corresponds to a distantly-supervised relation extraction task that constrains the logical forms produced by the semantic parser. This objective function is taken directly from Section 4.2.2.

A training example in the semantic objective consists of the set of sentences mentioning a pair of entities, $\mathbf{S}_{(e_1,e_2)} = \{s_1, s_2, ...\}$, paired with a binary vector representing the set of relations that the two entities participate in, $\mathbf{y}_{(e_1,e_2)}$. The distant supervision constraint Ψ forces the logical forms predicted for the sentences to entail the relations in $\mathbf{y}_{(e_1,e_2)}$. Ψ is a deterministic OR constraint that checks whether each logical form entails the relation instance $r(e_1, e_2)$, deterministically setting $y_r = 1$ if any logical form entails the instance and $y_r = 0$ otherwise.

Let $(\boldsymbol{\ell}, \boldsymbol{d}, \mathbf{t})$ represent a collection of semantic parses for the sentences $\mathbf{S}_{(e_1,e_2)}$. Let $\Gamma(\boldsymbol{\ell}, \boldsymbol{d}, \mathbf{t}, \mathbf{S}_{(e_1,e_2)}; \theta) = \sum_{i=1}^{|\mathbf{S}|} \Gamma(\ell_i, d_i, t_i, s_i; \theta)$ represent the total weight assigned by the parser to a collection of parses for the sentences $\mathbf{S}_{(e_1,e_2)}$. For the pair of entities (e_1, e_2), the semantic objective is:

$$O_{\text{sem}}(\theta) = |\max_{\hat{\boldsymbol{\ell}}, \hat{\boldsymbol{d}}, \hat{\mathbf{t}}} \Gamma(\hat{\boldsymbol{\ell}}, \hat{\boldsymbol{d}}, \hat{\mathbf{t}}, \mathbf{S}_{(e_1,e_2)}; \theta) - \max_{\boldsymbol{\ell}^*, \boldsymbol{d}^*, \mathbf{t}^*} \left(\Psi(\mathbf{y}_{(e_1,e_2)}, \boldsymbol{\ell}^*) + \Gamma(\boldsymbol{\ell}^*, \boldsymbol{d}^*, \mathbf{t}^*, \mathbf{S}_{(e_1,e_2)}; \theta)\right)|_+$$

Optimization

Training minimizes the joint objective using the structured perceptron algorithm, which can be viewed as the stochastic subgradient method (Ratliff et al., 2006) applied to the objective $O(\theta)$. The parameters are initialized to zero, i.e., $\theta^0 = 0$. Each iteration samples either a syntactic example (s_i, t_i) or a semantic example $(\mathbf{S}_{(e_1,e_2)}, \mathbf{y}_{(e_1,e_2)})$. If a syntactic example is sampled, the following parameter update is applied:

$$\begin{aligned}
\hat{\ell}, \hat{d}, \hat{t} &\leftarrow \arg\max_{\ell,d,t} \Gamma(\ell, d, t, s_i; \theta^t) \\
\ell^*, d^* &\leftarrow \arg\max_{\ell,d} \Gamma(\ell, d, t_i, s_i; \theta^t) \\
\theta^{t+1} &\leftarrow \theta^t + \phi(d^*, t_i, s_i) - \phi(\hat{d}, \hat{t}, s_i)
\end{aligned}$$

This update moves the parameters toward the features of the best parse with the correct syntactic derivation, $\phi(d^*, t_i, s_i)$. If a semantic example is sampled, the following update is applied instead:

$$\begin{aligned}
\hat{\boldsymbol{\ell}}, \hat{\boldsymbol{d}}, \hat{\mathbf{t}} &\leftarrow \arg\max_{\boldsymbol{\ell},\boldsymbol{d},\mathbf{t}} \Gamma(\boldsymbol{\ell}, \boldsymbol{d}, \mathbf{t}, \mathbf{S}_{(e_1,e_2)}; \theta^t) \\
\boldsymbol{\ell}^*, \boldsymbol{d}^*, \mathbf{t}^* &\leftarrow \arg\max_{\boldsymbol{\ell},\boldsymbol{d},\mathbf{t}} \Gamma(\boldsymbol{\ell}, \boldsymbol{d}, \mathbf{t}, \mathbf{S}_{(e_1,e_2)}; \theta^t) + \Psi(\mathbf{y}_{(e_1,e_2)}, \boldsymbol{\ell}) \\
\theta^{t+1} &\leftarrow \theta^t + \phi(\boldsymbol{d}^*, \mathbf{t}^*, \mathbf{S}_{(e_1,e_2)}) - \phi(\hat{\boldsymbol{d}}, \hat{\mathbf{t}}, \mathbf{S}_{(e_1,e_2)})
\end{aligned}$$

This update moves the parameters toward the features of the best set of parses that satisfy the distant supervision constraint. Training outputs the average of each iteration's parameters, $\bar{\theta} = \frac{1}{n}\sum_{t=1}^{n} \theta^t$. In practice, training performs only a single pass over the examples in the data set.

All of the maximizations above can be performed exactly using a CKY-style chart parsing algorithm except for the last one. This maximization is intractable due to the coupling between logical forms in $\boldsymbol{\ell}$ caused by enforcing the distant supervision constraint. The training procedure approximates this maximization in two steps. First, it performs a beam search to produce a list of candidate parses for each sentence $s \in \mathbf{S}_{(e_1,e_2)}$. It then extracts relation instances from each parse and applies the greedy inference algorithm from Hoffmann et al., (Hoffmann et al., 2011) to identify the best set of parses that satisfy the distant supervision constraint. This procedure skips any examples with sentences that cannot be parsed (due to beam search failures) or where the distant supervision constraint cannot be satisfied.

4.3.3 Evaluation

The evaluation measures ASP's syntactic and semantic parsing ability. The parser is trained on CCGbank and a corpus of Wikipedia sentences using NELL's predicate vocabulary. The syntactic analyses of the trained parser are evaluated against CCGbank, and its logical forms are evaluated on an information extraction task and against an annotated test set of Wikipedia sentences.

Data

The data sets for the evaluation consist of CCGbank, a corpus of dependency-parsed Wikipedia sentences, and a logical knowledge base derived from NELL and Freebase. Sections 02-21 of CCGbank were used for training, Section 00 for validation, and Section 23 for the final results. The knowledge base's predicate vocabulary is taken from NELL, and its instances are taken from Freebase using a manually-constructed mapping between Freebase and NELL. Using Freebase relation instances produces cleaner training data than NELL's automatically-extracted instances.

The relation instances and Wikipedia sentences were used to construct a data set for distantly-supervised relation extraction. This data set was constructed by identifying mentions of entities in each sentence using simple string matching, then aggregating these sentences by entity pair. 20% of the entity pairs were set aside for validation. To improve performance, only 3% of the entity pairs that do not participate in at least one relation were used for training. Furthermore, sentences containing more than 30 tokens were removed. The resulting training corpus contains 25k entity pairs (half of which participate in a relation), 41k sentences, and 71 distinct relation predicates.

Syntactic Evaluation

The syntactic evaluation measures ASP's ability to reproduce the predicate-argument dependencies in CCGbank. As in previous work, the evaluation uses *labeled* and *unlabeled* dependencies. Labeled dependencies are dependency structures with both words and semantic types removed, leaving two word indexes, a syntactic category, and an argument number. Unlabeled dependencies further eliminate the syntactic category and argument number, leaving a pair of word indexes. Performance is measured using precision, recall, and F-measure against the annotated dependency structures in CCGbank. Precision is the fraction of predicted dependencies which are in CCGbank, recall is the fraction of CCGbank dependencies produced by the parser, and F-measure is the harmonic mean of precision and recall.

The syntactic evaluation includes a purely syntactic version of ASP, called ASP-SYN, that is trained using only CCGbank. Comparing against this parser measures the effect on syntactic parsing performance of including the distantly-supervised relation extraction task during training.

Table 4.7 shows the results of the evaluation. For comparison, results for two existing syntactic CCG parsers are included: C&C (Clark and Curran, 2007a), and the system of Hockenmaier (Hockenmaier, 2003b). Both ASP and ASP-SYN perform reasonably well, within 2.5% of the performance of C&C at the same coverage level. However, ASP-SYN outperforms ASP by around 0.5%, suggesting that ASP's additional semantic knowledge slightly hurts syntactic parsing performance. This performance loss appears to be largely due to poor entity mention detection, as omitting entity mention lexicon entries at test time improves ASP's labeled and unlabeled F-scores by 0.3% on Section 00. The knowledge base contains many infrequently-mentioned entities with common names, such as the city "Of" in Turkey. These mentions contribute confusing semantic type information to the parser that can result in incorrect predictions. In this case, the type information would suggest that "of" is a proper noun. This problem is similar to the one noted in Section 4.2.3.

	Labeled Dependencies			Unlabeled Dependencies			
	P	R	F	P	R	F	Coverage
ASP	85.58	85.31	85.44	91.75	91.46	91.60	99.63
ASP-SYN	86.06	85.84	85.95	92.13	91.89	92.01	99.63
C&C (Clark and Curran, 2007a)	88.34	86.96	87.64	93.74	92.28	93.00	99.63
(Hockenmaier, 2003b)	84.3	84.6	84.4	91.8	92.2	92.0	99.83

Table 4.7: Syntactic parsing results for Section 23 of CCGbank. Parser performance is measured using precision (P), recall (R) and F-measure (F) of labeled and unlabeled dependencies.

Semantic Evaluation Baselines

The information extraction and annotated sentence evaluations compare ASP to two baselines. The first baseline, PIPELINE, is a pipelined syntax-then-semantics approach designed to mimic Boxer (Bos, 2005). This baseline first syntactically parses each sentence using ASP-SYN, then produces a semantic analysis by assigning a logical form to each word. This baseline is trained using the semantic objective (Section 4.3.2) while holding fixed the syntactic parse of each sentence. Note that, unlike Boxer, this baseline learns which logical form to assign to each word, and its logical forms contain NELL predicates.

The second baseline, K&M-2012, is the distantly-supervised approach from Section 4.2, representing the state-of-the-art in distantly-supervised semantic parsing. This approach trains a semantic parser by combining distant semantic supervision with syntactic supervision from dependency parses. The best performing variant of this system also uses dependency parses at test time to constrain the interpretation of test sentences – hence, this system also uses a pipelined syntax-then-semantics approach. To improve comparability, this approach was reimplemented using the ASP parsing model, which has richer features than were used in the previous implementation.

Information Extraction Evaluation

The information extraction evaluation uses each system to extract logical forms from a large corpus of sentences, then measures the fraction of extracted logical forms that are correct. The

Sentence	Extracted Logical Form
St. John, a Mexican-American born in San Francisco, California, her family comes from Zacatecas, Mexico.	$\lambda x.\exists y, z.\textsc{m}(x, \text{“st. john”}) \wedge \textsc{m}(y, \text{“san francisco”}) \wedge \textsc{personBornInLocation}(x, y) \wedge \textsc{cityLocatedInState}(y, z) \wedge \textsc{m}(z, \text{“california”})$
The capital and largest city of Laos is Vientiane and other major cities include Luang Prabang, Savannakhet and Pakse.	$\exists x, y.\textsc{m}(x, \text{“vientiane”}) \wedge \textsc{city}(x) \wedge \textsc{cityCapitalOfCountry}(x, y) \wedge \textsc{m}(y, \text{“laos”})$
Gellar next played a lead role in James Toback 's critically unsuccessful independent “Harvard Man” (2001), where she played the daughter of a mobster.	$\lambda x.\exists y.\textsc{m}(y, \text{“james toback”}) \wedge \textsc{directorDirectedMovie}(y, x) \wedge \textsc{m}(x, \text{“harvard man”})$

Figure 4.6: Logical forms produced by ASP for sentences in the information extraction corpus. Each logical form is extracted from the underlined sentence portion.

test set consists of 8.5k sentences sampled from the held-out Wikipedia sentences. Each system was run on this data set, extracting all logical forms from each sentence that entailed at least one category or relation instance. These extractions were ranked using the parser's inside chart score, and a random sample of 250 logical forms from each system were then manually annotated for correctness. Logical forms were marked correct if all category and relation instances entailed by the logical form were expressed by the sentence. Note that a correct logical form need not entail all of the relations expressed by the sentence, reflecting an emphasis on precision over recall. Figure 4.6 shows some example logical forms produced by ASP in the evaluation.

The annotated sample of logical forms allows us to estimate precision for each system as a function of the number of correct extractions (Figure 4.7). The number of correct extractions is directly proportional to recall and was estimated from the total number of extractions and precision at each rank in the sample. At low recall levels, all three systems have high precision, implying that their extracted logical forms express facts found in the sentence. However, ASP produces 3 times more correct logical forms than either baseline because it jointly analyzes syntax and semantics. The baselines suffer from reduced recall because they depend on receiving an accurate syntactic parse as input; syntactic parsing errors cause these systems to fail.

Examining the incorrect logical forms produced by ASP reveals that incorrect mention detection is by far the most common source of mistakes. Approximately 50% of errors are caused by marking common nouns as entity mentions (e.g., marking “coin” as a COMPANY). These errors

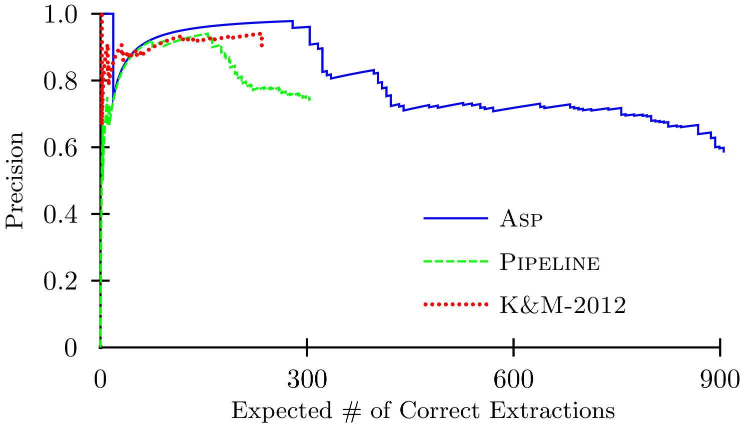


Figure 4.7: Logical form precision as a function of the expected number of correct extracted logical forms. ASP extracts more correct logical forms because it jointly analyzes syntax and semantics.

occur because the knowledge base contains many infrequently mentioned entities with relatively common names. Another 30% of errors are caused by assigning an incorrect type to a common proper noun (e.g, marking "Bolivia" as a CITY). This analysis suggests that performing entity linking before parsing could significantly reduce errors.

Annotated Sentence Evaluation

A limitation of the previous evaluation is that it does not measure the completeness of predicted logical forms, nor estimate what portion of sentences are left unanalyzed. This second evaluation measures these quantities.

The data for this evaluation consists of sentences annotated with logical forms for subspans. Wikipedia sentences from the held-out set were manually annotated with logical forms for the largest subspans for which a logical form existed. To avoid trivial cases, every annotated logical forms contains at least one category or relation predicate and at least one mention. The annotated logical forms also do not contain mentions of entities that are not in the knowledge base, as no system would be able to correctly identify these mentions. The corpus contains 97 sentences with 100 annotated logical forms.

	Logical Form Accuracy	Extraction Precision	Extraction Recall
ASP	0.28	0.90	0.32
K&M-2012	0.14	1.00	0.06
PIPELINE	0.2	0.63	0.17

Table 4.8: Logical form accuracy and extraction precision/recall on the annotated test set. The high extraction recall for ASP shows that it produces more complete logical forms than either baseline.

Performance on this data set was measured using two metrics: logical form accuracy, and extraction precision/recall. Logical form accuracy examines the predicted logical form for the smallest subspan of the sentence that contains the annotated span, and marks this prediction correct if it exactly matches the annotation. A limitation of this metric is that it does not assign partial credit to logical forms that are close to, but do not exactly match, the annotation. The extraction metric assigns partial credit by computing the precision and recall of the category and relation instances entailed by the predicted logical form, using the instances entailed by the annotated logical form as the gold standard. Figure 4.8 shows the computation of both error metrics on two examples from the test corpus.

Table 4.8 shows the results of the annotated sentence evaluation. ASP outperforms both baselines in logical form accuracy and extraction recall, demonstrating that it produces more complete analyses than either baseline. The extraction precision of 90% suggests that ASP rarely includes incorrect information in its predicted logical forms. However, the recall of 32% suggests that semantic information present in the sentence is often not included in the predicted logical form. Note that precision in this evaluation is higher than precision in Section 4.3.3 because every sentence in the data set has a correct logical form. Many of the precision errors in the previous evaluation occur because the parser predicts a logical form for sentences for which no correct logical form exists.

Sentence: *De Niro and Joe Pesci in "Goodfellas"* offered a virtuoso display of the director's bravura cinematic technique and reestablished, enhanced, and consolidated his reputation.
Annotation:
LF: $\lambda x.\forall p \in \{\lambda d.\textsc{m}(d, \text{"de niro"}), \lambda j.\textsc{m}(j, \text{"joe pesci"})\} \exists y.p(x) \wedge \textsc{starredInMovie}(x, y) \wedge \textsc{m}(y, \text{"goodfellas"})$
Instances: $\textsc{starredInMovie}$(de niro, goodfellas), $\textsc{starredInMovie}$(joe pesci, goodfellas)
Prediction:
LF: $\lambda x.\forall p \in \{\lambda d.\textsc{m}(d, \text{"de niro"}), \lambda j.\textsc{m}(j, \text{"joe pesci"})\} \exists y.p(x) \wedge \textsc{starredInMovie}(x, y) \wedge \textsc{m}(y, \text{"goodfellas"})$
Instances: $\textsc{starredInMovie}$(de niro, goodfellas), $\textsc{starredInMovie}$(joe pesci, goodfellas)
Logical form accuracy: 1 / 1 **Extraction Precision:** 2 / 2 **Extraction Recall:** 2 / 2

Sentence: In addition to the University of Illinois, *Champaign is also home to Parkland College*.
Annotation:
LF: $\exists c, p.\textsc{m}(c, \text{"champaign"}) \wedge \textsc{city}(c) \wedge \textsc{m}(p, \text{"parkland college"}) \wedge \textsc{universityInCity}(p, c)$
Instances: $\textsc{city}$(champaign), $\textsc{universityInCity}$(parkland college, champaign)
Prediction:
LF 1: $\lambda x.\exists y \textsc{m}(y, \text{"illinois"}) \wedge \textsc{m}(x, \text{"university"}) \wedge \textsc{cityLocatedInState}(x, y)$
LF 2: $\exists c, p.\textsc{m}(c, \text{"champaign"}) \wedge \textsc{city}(c) \wedge \textsc{m}(p, \text{"parkland college"}) \wedge \textsc{universityInCity}(p, c)$
Instances: $\textsc{city}$(champaign), $\textsc{universityInCity}$(parkland college, champaign),
$\textsc{cityLocatedInState}$(university, illinois)
Logical form accuracy: 1 / 1 **Extraction Precision:** 2 / 3 **Extraction Recall:** 2 / 2

Figure 4.8: Two test examples with ASP's predictions and error calculations. The annotated logical forms are for the italicized sentence spans, while the extracted logical forms are for the underlined spans.

4.4 Discussion

This chapter has presented two algorithms for training semantic parsers using distant supervision. The key innovation used by both algorithms is a distant supervision constraint that trains a semantic parser to extract the relations instances in a knowledge base from a large corpus of unlabeled sentences. However, this semantic constraint alone is not sufficient – syntactic information was shown to play a key role in producing accurate semantic parses. The two algorithms presented here differ in their source of syntactic information. The first algorithm used a simple semantic parser and constrained its parses to agree with dependency parses. The second algorithm trained a joint syntactic and semantic parser using annotated syntactic trees from CCGbank.

The experimental results in this chapter demonstrated three major results. First, semantic parsers trained with distant supervision are effective relation extraction and question answering systems. Second, syntactic information is necessary to produce accurate semantic parses of complex text, such as sentences from the web. Third, jointly inferring the syntactic and semantic structure of a sentence enables a semantic parser to produce more accurate semantic parses than pipelined syntax-then-semantics approaches.

The positioning of this chapter according to the axes of variation is as follows:

1. **Learning the knowledge base** – The knowledge base is given to the learning algorithm and not automatically learned.

2. **Predicate vocabulary** – The semantic parsers use the closed predicate vocabulary of the given knowledge base.

3. **Semantic parser supervision** – Unlabeled sentences from the web and a knowledge base are used to train the semantic parser.

4. **Syntactic information** – Syntactic information was found to be very helpful in producing accurate semantic parses. Syntactic information was provided in the form of either dependency parses or syntactic CCG parses.

One lesson from this chapter is the limitations of language understanding system that relies of a given, fixed knowledge base. Despite using a reasonably large subset of NELL (∼130 predicates), the parsers in this chapter can produce logical forms for only a tiny fraction of sentences. For example, given a corpus of 8500 sentences, ASP produces logical forms for around 900 sentences. Note that these logical forms are partial analyses, often entailing only a single predicate instance. The limited coverage is largely a function of the knowledge base, which does not contain the predicates necessary to semantically analyze most language. There are also many cases where the mapping from language to the knowledge base is inexact; for example, should the word "town" be mapped to the predicate CITY? These observations suggest that it is necessary to open the predicate vocabulary and learn the knowledge base to obtain broad coverage semantic analyses. The next chapter explores training language understanding systems with an open predicate vocabulary.

This chapter also suggests that syntactically-parsed, but otherwise unlabeled, text is an effective source of supervision for training semantic parsers. The next chapter further explores the use of such syntactically-parsed text as a source of supervision for training language understanding systems.

Chapter 5

Semantic Parsing with an Open Predicate Vocabulary

Abstract

The approaches described in the previous chapter trained a semantic parser to map text to logical forms containing predicates from a fixed knowledge base. A fundamental limitation of these approaches is that their coverage is restricted by the closed predicate vocabulary of the knowledge base. This chapter describes a new approach to building a language understanding system that uses an *open* predicate vocabulary, enabling it to produce denotations for phrases such as “Republican front-runner from Texas” whose semantics could not be represented using the knowledge bases of the previous chapter. The system’s overall design is the opposite of the previously-described approaches: it uses an unlearned, deterministic semantic parser, and instead learns a distribution over knowledge bases. The semantic parser is given by a collection of rules that converts a sentence’s syntactic CCG parse into a logical form containing predicates derived from the words in the sentence. Instead of a fixed knowledge base, the system has a learned probabilistic database representing a distribution over knowledge bases. This probabilistic database is trained using a corpus of entity-linked text and probabilistic matrix factorization with a novel ranking objective function. An evaluation on a compositional question answering task demonstrates that this open vocabulary approach to language understanding outperforms several competitive baselines and that it can correctly answer many questions that cannot be answered by querying Freebase.

The traditional language understanding paradigm assumes that world knowledge can be encoded using a closed vocabulary of formal predicates. In this paradigm, the understanding system is given a knowledge base as input, and the language understanding problem reduces to semantic parsing (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005), i.e., mapping from text to logical forms containing predicates from the given knowledge base. This paradigm has been successfully used in applications such as answering questions using a large knowledge base (Berant and Liang, 2014; Berant et al., 2013; Cai and Yates, 2013b; Krishnamurthy and Mitchell, 2012; Kwiatkowski et al., 2013; Reddy et al., 2014a; Yahya et al., 2012) (Chapter 4).

However, the closed predicate vocabulary assumed by the traditional language understanding paradigm has inherent limitations. First, a closed predicate vocabulary will have limited coverage, as such vocabularies are typically manually constructed. The semantic evaluations in Section 4.3.3 highlight this coverage problem: no logical form could be produced for most sentences, and, when a logical form was produced, it represented the meaning of only a small portion of the sentence. Second, a closed predicate vocabulary may abstract away potentially relevant semantic differences. For example, the semantics of “Republican front-runner” cannot be adequately encoded using Freebase predicates because it lacks the concept of a “front-runner.” This concept could be encoded as “politician” at the cost of abstracting away the distinction between the two. As this example illustrates, these two problems are prevalent in even the largest knowledge bases.

This chapter considers the problem of building a language understanding system with an *open* predicate vocabulary. In an open predicate vocabulary, each natural language word or phrase is given its own formal predicate (Banko et al., 2007; Riedel et al., 2013), thereby permitting the system to capture subtle semantic distinctions and achieve high coverage. Using an open predicate vocabulary considerably changes the design of a language understanding system. First, it forces the system to *learn* its knowledge base, as the system no longer receives a knowledge base as input. Second, it simplifies the problem of semantic parsing, as it is no longer neces-

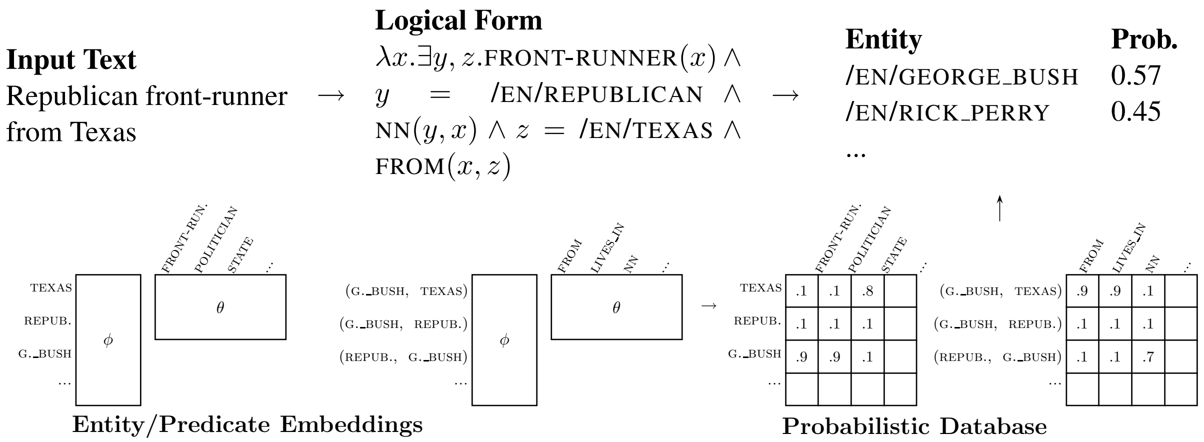

Input Text
Republican front-runner from Texas
→
Logical Form
$\lambda x.\exists y, z.\text{FRONT-RUNNER}(x) \wedge y = \text{/EN/REPUBLICAN} \wedge \text{NN}(y, x) \wedge z = \text{/EN/TEXAS} \wedge \text{FROM}(x, z)$
→
Entity
Prob.
/EN/GEORGE_BUSH 0.57
/EN/RICK_PERRY 0.45
...
↑
FRONT-RUN.
POLITICIAN
STATE
...
TEXAS
REPUB.
G._BUSH
...
φ
θ
FROM
LIVES_IN
NN
...
(G._BUSH, TEXAS)
(G._BUSH, REPUB.)
(REPUB., G._BUSH)
...
φ
θ
→
TEXAS .1 .1 .8
REPUB. .1 .1 .1
G._BUSH .9 .9 .1
...
(G._BUSH, TEXAS) .9 .9 .1
(G._BUSH, REPUB.) .1 .1 .1
(REPUB., G._BUSH) .1 .1 .7
...
Entity/Predicate Embeddings
Probabilistic Database

Figure 5.1: Overview of the language understanding system in this chapter. Top left: the text is converted to logical form by CCG syntactic parsing and a collection of manually-defined rules. Bottom: low-dimensional embeddings of each entity (entity pair) and category (relation) are learned from an entity-linked web corpus. These embeddings are used to construct a probabilistic database. The labels of these matrices are shortened for space reasons. Top right: evaluating the logical form on the probabilistic database computes the marginal probability that each entity is an element of the text's denotation.

sary to map between words and knowledge base symbols. The language understanding system in this chapter is therefore quite different from those of the previous chapter, even though its input/output specification is similar. The system can still predict the denotation of a text in terms of knowledge base entities, but this prediction is based on an unlearned, deterministic semantic parser, and a learned knowledge base.

The language understanding system in this chapter is a probabilistic model that defines a distribution over denotations (sets of Freebase entities) given an input text. The model has two components. The first component is a rule-based semantic parser that uses a syntactic CCG parser and manually-defined rules to map entity-linked texts to logical forms containing predicates derived from the words in the text (Figure 5.1, top left). The second component is a probabilistic database with a possible worlds semantics that defines a distribution over denotations for each textually-derived predicate. This database can be viewed as a distribution over knowledge bases that assigns an independent truth probability to each predicate instance, such as $P(\text{FRONT-RUNNER}(\text{/EN/GEORGE\_BUSH})) = 0.9$ (Figure 5.1, bottom right). The probabilities in the database are computed from learned entity and predicate embeddings (Figure 5.1, bottom left). Together, the semantic parser and probabilistic database define an exponentially-large distribution over denotations for an input text; to simplify this output, inference in the model computes the marginal probability, over all possible knowledge bases, that each entity is an element of the text's denotation (Figure 5.1, top right).

The learning problem in this approach is to train the probabilistic database to predict a denotation for each predicate. This problem is posed as probabilistic matrix factorization with a novel query/answer ranking objective. This factorization learns a low-dimensional embedding of each entity (entity pair) and category (relation) such that the denotation of a predicate is likely to contain entities or entity pairs with nearby vectors. The database is trained using a collection of logical form queries with observed entity answers that is automatically generated by analyzing entity-linked text with the rule-based semantic parser. The query/answer ranking objective, when optimized, trains the database to rank the observed answers for each query above unobserved answers.

This chapter further presents an empirical evaluation of the trained open predicate vocabulary language understanding system on a compositional question answering task. The evaluation demonstrates that this system outperforms several baselines, and that the new training objective improves performance over a previously-proposed objective. The evaluation also examines the trade-offs between open and closed predicate vocabularies by comparing against an annotated Freebase query for each question. This comparison reveals that, when Freebase covers the question, it returns high precision answers. However, many of the questions not covered by Freebase can be correctly answered using the open vocabulary approach.

The contents of this chapter are derived from Krishnamurthy and Mitchell (2015).

5.1 System Overview

The system in this chapter aims to predict a denotation γ for a given natural language text s. The denotation γ is the set of Freebase entities that s refers to; for example, if $s =$ "president of the US," then $\gamma = \{\text{/EN/OBAMA}, \text{/EN/BUSH}, ...\}$.[1] This prediction problem can be represented using the following probabilistic model:

$$P(\gamma|s) = \sum_K \sum_\ell P(\gamma|\ell, K)P(K)P(\ell|s)$$

The first term in this factorization, $P(\ell|s)$, is a distribution over logical forms ℓ given the text s. This term corresponds to the rule-based semantic parser (Section 5.2). This semantic parser is deterministic, so this term assigns probability 1 to a single logical form for each text. The second term, $P(K)$, represents a distribution over knowledge bases, where each knowledge base is an assignment of truth values to all possible predicate instances. This distribution is represented by a probabilistic database (Section 5.3). The final term, $P(\gamma|\ell, K)$, deterministically evaluates the logical form ℓ on the knowledge base K to produce a denotation γ. This term represents query evaluation against a fixed database as in other work on semantic parsing.

Section 5.4 describes inference in this model. To produce a ranked list of entities (Figure 5.1, top right) from $P(\gamma|s)$, the system computes the marginal probability, over all possible knowledge bases, that each entity is an element of the denotation γ. This problem corresponds to query evaluation in a probabilistic database, which is known to be tractable in many cases (Suciu et al., 2011).

Section 5.5 describes training, which estimates parameters for the probabilistic database $P(K)$. This step first automatically generates training data using the rule-based semantic parser. This data is used to formulate a matrix factorization problem that is optimized to estimate the

[1] This chapter uses a simple model-theoretic semantics where the denotation of a noun phrase is a set of entities and the denotation of a sentence is either true or false. However, for notational convenience, denotations γ will be treated as sets of entities throughout.

database parameters.

5.2 Rule-Based Semantic Parser

The first part of the language understanding system is a rule-based system that deterministically computes a logical form for a text. This component is used during inference to analyze the logical structure of text, and during training to generate training data (see Section 5.5.1). Several input/output pairs for this system are shown in Figure 5.2.

The conversion to logical form has 3 phases:

1. **CCG syntactic parsing** parses the text and applies several deterministic syntactic transformations to facilitate semantic analysis.
2. **Entity linking** marks known Freebase entities in the text.
3. **Semantic analysis** assigns a logical form to each word, then composes them to produce a logical form for the complete text.

5.2.1 Syntactic Parsing

Syntactic parsing uses the ASP-SYN parser (Section 4.3.3) to produce a syntactic CCG parse of the text. It then applies several deterministic transformations to make the syntactic parse more amenable to semantic analysis. One transformation marks NPs in conjunctions by replacing their syntactic category with NP[conj]. This transformation allows semantic analysis to distinguish between appositives and comma-separated lists. Another transformation makes all verb arguments core arguments, i.e., using the category PP/NP as opposed to $((S\backslash NP)\backslash(S\backslash NP))/NP$. This step simplifies the semantic analysis of verbs with prepositional phrase arguments. The final transformation adds a word feature to each PP category, e.g., mapping PP to PP[by]. These features are used to generate verb-preposition relation predicates, such as DIRECTED_BY.

Dan Hesse, CEO of Sprint
$\lambda x.\exists y.x = \text{/EN/DAN\_HESSE} \wedge \text{CEO}(x) \wedge \text{OF}(x, y) \wedge y = \text{/EN/SPRINT}$

Yankees pitcher
$\lambda x.\exists y.\text{PITCHER}(x) \wedge \text{NN}(y, x) \wedge y = \text{/EN/YANKEES}$

Tom Cruise plays Maverick in the movie Top Gun.
$\exists x, y, z.x = \text{/EN/TOM\_CRUISE} \wedge \text{PLAYS}(x, y) \wedge y = \text{/EN/MAVERICK\_(CHARACTER)} \wedge \text{PLAYS\_IN}(x, z) \wedge z = \text{/EN/TOP\_GUN}$

Figure 5.2: Example input/output pairs for our semantic analysis system. Mentions of Freebase entities in the text are indicated by underlines.

5.2.2 Entity Linking

The second step is to identify mentions of Freebase entities in the text. This step could be performed by an off-the-shelf entity linking system (Milne and Witten, 2008; Ratinov et al., 2011) or string matching. However, this chapter uses the Clueweb 2009 text corpus and the automatically generated entity linking provided in (Gabrilovich et al., 2013).

The semantic analysis system incorporates the provided entity links into the syntactic parse provided that they are consistent with the parse structure. Specifically, it requires that each mention is either (1) a constituent in the parse tree with syntactic category N or NP or (2) a collection of N/N or NP/NP modifiers with a single head word. The first case covers noun and noun phrase mentions, while the second case covers noun compounds. In both cases, a single multi-word terminal spanning the mention is substituted into the parse tree and special semantic rules for mentions are invoked during the next step.

5.2.3 Semantic Analysis

The final step uses the syntactic parse and entity links to produce a logical form for the text. This step induces a logical form for every word in the text based on its syntactic CCG category. Composing these logical forms according to the syntactic parse produces a logical form for the entire text.

The semantic analyses of this component are based on a relatively naïve model-theoretic semantics. They focus on language whose semantics can be represented with unary and binary predicates, ignoring, for example, temporal scope and superlatives. Generally, the system models nouns and adjectives as unary predicates, and verbs and prepositions as binary predicates. Special multi-word predicates are generated for verb-preposition combinations. Entity mentions are mapped to the mentioned entity in the logical form. The system also includes special rules for analyzing conjunctions, appositives, relativizing conjunctions and negation. A list of the most important templates used to produce these logical forms is listed in Table 5.1.

There are several notable aspects of this component's design. First, multi-argument verbs are analyzed using pairwise relations, as in the third example in Figure 5.2. This analysis allows the system to avoid reasoning about entity triples (quadruples, etc.), which are challenging to model with a probabilistic database due to sparsity. Second, noun-preposition combinations are analyzed as a category and relation, as in the first example in Figure 5.2. Preliminary experiments found that creating a combined predicate for the noun and preposition results in worse performance because it dramatically increases the sparsity of training instances for the combined relations. Third, entity mentions with the N/N category are analyzed using a special noun-noun relation, as in the second example in Figure 5.2. This analysis allows the matrix factorization to learn that the noun-noun relation shares instances with other relations, for example, to learn that "city in Texas" implies "Texan city." Predicate names are created from the lowercased word with no lemmatization or other normalization.

5.2.4 Discussion

The scope of this semantic analysis system is somewhat limited relative to other similar systems (Bos, 2008; Lewis and Steedman, 2013) as it only outputs existentially-quantified conjunctions of predicates. The goal in building this system was to analyze noun phrases and simple sentences, for which this representation generally suffices. The reason for this focus is twofold. First, this subset of language is sufficient to capture much of the language surrounding Freebase entities.

Words	Syntactic Categories	Logical Form Template
*	$(NP\backslash NP)/NP$	$\lambda f.\lambda g.\lambda x.g(x) \wedge \exists y.f(y) \wedge word(x,y)$
*	$(S\backslash NP)$	$\lambda g.\exists x.g(x) \wedge word(x)$
*	$(S\backslash NP)/NP$	$\lambda f.\lambda g.\exists x.g(x) \wedge \exists y.f(y) \wedge word(x,y)$
*	$(S\backslash NP)/PP[2]$	$\lambda f.\lambda g.\exists x.g(x) \wedge \exists y.f(y) \wedge word + pp_2(x,y)$
*	$((S\backslash NP)/PP[2])/NP$	$\lambda h.\lambda f.\lambda g.\exists x.g(x) \wedge \exists y.f(y) \wedge word + pp_2(x,y) \wedge \exists z.h(z) \wedge word(x,z)$
*	$(S\backslash NP)/(S\backslash NP)$	$\lambda f.f$
*	$(NP/N)\backslash NP$	$\lambda f.\lambda g.\lambda x.g(x) \wedge \exists y.f(y) \wedge word(y,x)$
*	NP/N	$\lambda f.f$
*	PP/NP	$\lambda f.f$
*	N/N	$\lambda f.\lambda x.word(x) \wedge f(x)$
*	$(N/N)/(N/N)$	$\lambda f.\lambda g.f(g)$
*	$N\backslash N$	$\lambda f.\lambda x.word(x) \wedge f(x)$
*	N	$\lambda x.word(x)$
*	NP	$\lambda x.word(x)$
*	$NP/PP[1]$	$\lambda f.\lambda x.\exists y.word(x) \wedge word + pp_1(x,y) \wedge f(y)$
*	$(NP/(S\backslash NP))$	$\lambda f.\lambda y.f(\lambda x.x = y)$
*	$(NP/(S/NP))$	$\lambda f.\lambda y.f(\lambda x.x = y)$
*	$((NP\backslash NP)/(S\backslash NP))$	$\lambda f.\lambda g.\lambda x.g(x) \wedge f(\lambda y.y = x)$
is was are	$(S\backslash NP)/NP$	$\lambda f.\lambda g.\exists x.g(x) \wedge f(x)$
not	(N/N)	$\lambda f.\lambda x.\neg f(x)$
not	$(S\backslash NP)/(S\backslash NP)$	$\lambda f.\lambda g.\exists x.g(x) \wedge \neg f(\lambda y.y = x)$

Table 5.1: Partial listing of logical form templates used to perform the rule-based semantic analysis. Each rule matches a syntactic category and, optionally, a specific list of words. Matched rules instantiate the logical forms on the right, substituting the lowercased matched word for $word$. PP arguments are uniquely numbered and identified with prepositions in the sentence; these prepositions are substituted for each pp_i expression. Templates for verbs with more arguments have been omitted, but each additional argument operates analogously.

Second, for various technical reasons, this restricted semantic representation is easier to use (and more informative) for training the probabilistic database (see Section 5.5.3).

Note that this system can be straightforwardly extended to model additional linguistic phenomena, such as additional logical operators and generalized quantifiers, by writing additional rules. The semantics of logical forms including these operations are well-defined in the model, and the system does not even need to be re-trained to incorporate these additions.

5.3 Probabilistic Database

The second part of the language understanding model is a probabilistic database. This database represents a distribution over knowledge bases, where each knowledge base is an assignment of truth values to every predicate instance. Equivalently, the probabilistic database can be viewed as a distribution over databases.

Formally, a probabilistic database is a collection of random variables, each of which represents the truth value of a single predicate instance. Given entities $e \in E$, categories $c \in C$, and relations $r \in R$ the probabilistic database contains boolean random variables $c(e)$ and $r(e_1, e_2)$ for each category and relation instance, respectively. All of these random variables are assumed to be independent. Let a knowledge base K represent an assignment of truth values to all of these random variables, where $c(e) = K_{c,e}$ and $r(e_1, e_2) = K_{r,e_1,e_2}$. By independence, the probability of a knowledge base can be written as:

$$\begin{aligned} P(K) \;&=\; \prod_{e \in E} \prod_{c \in C} P(c(e) = K_{c,e}) \times \\ &\quad\; \prod_{e_1 \in E} \prod_{e_2 \in E} \prod_{r \in R} P(r(e_1, e_2) = K_{r,e_1,e_2}) \end{aligned}$$

The next section discusses how probabilistic matrix factorization is used to model the probabilities of these predicate instances.

5.3.1 Matrix Factorization Model

The probabilistic matrix factorization model treats the truth of each predicate instance as an independent boolean random variable that is true with probability:

$$\begin{aligned} P(c(e) = \textsc{True}) &= \sigma(\theta_c^T \phi_e) \\ P(r(e_1, e_2) = \textsc{True}) &= \sigma(\theta_r^T \phi_{(e_1,e_2)}) \end{aligned}$$

Above, $\sigma(x) = \frac{e^x}{1+e^x}$ is the logistic function. In these equations, θ_c and θ_r represent k-dimensional vectors of per-predicate parameters, while ϕ_e and $\phi_{(e_1,e_2)}$ represent k-dimensional vector embeddings of each entity and entity pair. This model contains a low-dimensional embedding of each predicate and entity such that each predicate's denotation has a high probability of containing entities with nearby vectors. The probability that each variable is false is simply 1 minus the probability that it is true.

This model can be viewed as matrix factorization, as depicted in Figure 5.1. The category and relation instance probabilities can be arranged in a pair of matrices of dimension $|E| \times |C|$ and $|E|^2 \times |R|$. Each row of these matrices represents an entity or entity pair, each column represents a category or relation, and each value is between 0 and 1 and represents a truth probability (Figure 5.1, bottom right). These two matrices are factored into matrices of size $|E| \times k$ and $k \times |C|$, and $|E|^2 \times k$ and $k \times |R|$, respectively, containing k-dimensional embeddings of each entity, category, entity pair and relation (Figure 5.1, bottom left). These low-dimensional embeddings are represented by the parameters ϕ and θ.

5.4 Inference: Computing Marginal Probabilities

Inference computes the marginal probability, over all possible knowledge bases, that each entity is an element of a text's denotation. In many cases – depending on the text – these marginal probabilities can be computed exactly in polynomial time.

The inference problem is to calculate $P(e \in \gamma | s)$ for each entity e. Because both the semantic parser $P(\ell|s)$ and query evaluation $P(\gamma|\ell, K)$ are deterministic, this problem can be rewritten as:

$$\begin{aligned} P(e \in \gamma | s) &= \sum_{\gamma} \mathbf{1}(e \in \gamma) P(\gamma | s) \\ &= \sum_{K} \mathbf{1}(e \in [\![\ell]\!]_K) P(K) \end{aligned}$$

Above, ℓ represents the logical form for the text s produced by the rule-based semantic parser, and **1** represents the indicator function. The notation $[\![\ell]\!]_K$ represents denotation produced by (deterministically) evaluating the logical form ℓ on the knowledge base K. This inference problem corresponds to query evaluation in a probabilistic database, which is #P-hard in general. Intuitively, this problem can be difficult because $P(\gamma|s)$ is a joint distribution over sets of entities that can be exponentially large in the number of entities.

However, a large subset of probabilistic database queries, known as *safe* queries, permit polynomial time evaluation (Dalvi and Suciu, 2007). Safe queries can be evaluated extensionally using a probabilistic notion of a denotation that treats each entity as independent. Let $[\![\ell]\!]_P$ denote a probabilistic denotation, which is a function from entities (or entity pairs) to probabilities, i.e., $[\![\ell]\!]_P(e) \in [0, 1]$. The denotation of a logical form is then computed recursively, in the same manner as a non-probabilistic denotation, using probabilistic extensions of the typical rules, such as:

$$\begin{aligned} [\![c]\!]_P(e) &= \sum_K P(K) \mathbf{1}(K_{c,e}) \\ [\![r]\!]_P(e_1, e_2) &= \sum_K P(K) \mathbf{1}(K_{r,e_1,e_2}) \\ [\![c_1(x) \wedge c_2(x)]\!]_P(e) &= [\![c_1]\!]_P(e) \times [\![c_2]\!]_P(e) \\ [\![\exists y . r(x,y)]\!]_P(e) &= 1 - \prod_{y \in E} (1 - [\![r]\!]_P(e, y)) \end{aligned}$$

The first two rules are base cases that simply retrieve predicate probabilities from the probabilistic database. The remaining rules compute the probabilistic denotation of a logical form from the denotations of its parts.[2] The formula for the probabilistic computation on the right of each of these rules is a straightforward consequence of the (assumed) independence of entities. For example, the last rule computes the probability of an OR of a set of independent random variables (indexed by y) using the identity $A \vee B = \neg(\neg A \wedge \neg B)$. For safe queries, $[\![\ell]\!]_P(e) = P(e \in \gamma|s)$, that is, the probabilistic denotation computed according to the above rules *is equal to* the marginal probability distribution. In practice, all of the queries in the experiments are safe, because they contain only one query variable and do not contain repeated predicates. For more information on query evaluation in probabilistic databases, I refer the reader to (Suciu et al., 2011).

Note that inference does not compute the most probable denotation, $\max_\gamma P(\gamma|s)$. In some sense, the most probable denotation is the correct output for a model-theoretic semantics. However, it is highly sensitive to the probabilities in the database, and in many cases it is empty (because a conjunction of independent boolean random variables is unlikely to be true). Producing a ranked list of entities is also useful for evaluation purposes.

5.5 Training

The goal of training is to estimate the parameters θ and ϕ for the probabilistic database. This section presents two different objective functions for estimating these parameters that use slightly different forms of training data. The first objective is based on prior work and aims to rank observed predicate instances above unobserved instances. The second objective is a novel query ranking objective that aims to rank observed answers to queries above unobserved answers. In both cases, training has two phases. The first step is to generate training data, in the form of either observed assertions or query-answer pairs, by applying the rule-based semantic analysis system to a corpus of entity-linked web text. The second step is to optimize the parameters of

[2] This listing of rules is partial as it does not include, e.g., negation or joins between one-argument and two-argument logical forms. However, the corresponding rules are easy to derive.

Original sentence and logical form
General Powell, appearing Sunday on CNN 's Late Edition, said ...
$\exists w, x, y, z.\quad w = /\textsc{en/powell} \wedge \textsc{general}(w) \wedge \textsc{app.}(w, x) \wedge \textsc{sunday}(x) \wedge \textsc{app.\_on}(w, y) \wedge y = /\textsc{en/late} \wedge \textsc{'s}(z, y) \wedge z = /\textsc{en/cnn} \wedge \textsc{said}(w, ...)$

Simplified logical form
$\exists w, y, z.\ w = /\textsc{en/powell} \wedge \textsc{general}(w) \wedge \textsc{app.\_on}(w, y) \wedge y = /\textsc{en/late} \wedge \textsc{'s}(z, y) \wedge z = /\textsc{en/cnn}$

Instances	**Queries**	**Answers**
$\textsc{general}(/\textsc{en/powell})$	$\lambda w.\textsc{general}(w) \wedge \textsc{app.\_on}(w, /\textsc{en/late})$	$/\textsc{en/powell}$
$\textsc{app.\_on}(/\textsc{en/powell}, /\textsc{en/late})$	$\lambda y.\textsc{app.\_on}(/\textsc{en/powell}, y) \wedge \textsc{'s}(/\textsc{en/cnn}, y)$	$/\textsc{en/late}$
$\textsc{'s}(/\textsc{en/cnn}, /\textsc{en/late})$	$\lambda z.\textsc{'s}(z, /\textsc{en/late})$	$/\textsc{en/cnn}$

Figure 5.3: Illustration of training data generation applied to a single sentence. Two types of training data are generated, predicate instances and queries with observed answers, by semantically analyzing the sentence and extracting portions of the generated logical form with observed entity arguments. The predicate instances are extracted from the conjuncts in the simplified logical form, and the queries are created by removing a single entity from the simplified logical form.

the probabilistic database to rank observed assertions or answers above unobserved assertions or answers.

5.5.1 Training Data

Training data is generated by applying the process illustrated in Figure 5.3 to each sentence in an entity-linked web corpus. First, this process applies the rule-based semantic parser to the sentence to produce a logical form. Next, it extracts portions of this logical form where every variable is bound to a particular Freebase entity, resulting in a simplified logical form. Because the logical forms are existentially-quantified conjunctions of predicates, this step simply discards any conjuncts in the logical form containing a variable that is not bound to a Freebase entity. From this simplified logical form, it generates two types of training data: (1) predicate instances, and (2) queries with known answers (see Figure 5.3). In both cases, the corpus consists entirely of assumed-true statements or queries with assumed-correct answers, making obtaining negative examples a major challenge for training.

5.5.2 Predicate Ranking Objective

Riedel et al. (Riedel et al., 2013) introduced a ranking objective to work around the lack of negative examples in a similar matrix factorization problem. Their objective is a modified version of Bayesian Personalized Ranking (Rendle et al., 2009) that aims to rank observed predicate instances above unobserved instances.

The predicate ranking objective O_P uses true predicate instances (Figure 5.3, bottom left) as training data. This data consists of two collections, $\{(c_i, e_i)\}_{i=1}^{n}$ and $\{(r_j, e_{j1}, e_{j2})\}_{j=1}^{m}$, of observed category and relation instances $c_i(e_i)$ and $r_j(e_{j1}, e_{j2})$. This objective is:

$$O_P(\theta, \phi) \;=\; \sum_{i=1}^{n} \log \sigma(\theta_{c_i}^T(\phi_{e_i} - \phi_{e'_i})) + \sum_{j=1}^{m} \log \sigma(\theta_{r_j}^T(\phi_{(e_{j1},e_{j2})} - \phi_{(e'_{j1},e'_{j2})}))$$

where e'_i is a randomly sampled entity such that (c_i, e'_i) does not occur in the training data. Similarly, (e'_{j1}, e'_{j2}) is a random entity tuple such that (r_j, e'_{j1}, e'_{j2}) does not occur. Maximizing this function attempts to find θ_{c_i}, ϕ_{e_i} and $\phi_{e'_i}$ such that $P(c_i(e_i))$ is larger than $P(c_i(e'_i))$. During training, e'_i and (e'_{j1}, e'_{j2}) are resampled on each pass over the data set according to the empirical frequency of each entity or tuple.

5.5.3 Query Ranking Objective

The previous objective aims to rank the entities within each predicate well. However, such within-predicate rankings are insufficient to produce correct answers for queries containing multiple predicates – the scores for each predicate must further be calibrated to work well with each other given the independence assumptions of the probabilistic database. Calibration is necessary because the probability that an entity is in the denotation of a multiple predicate query, such as $\lambda x.\text{PRESIDENT}(x) \wedge \text{NN}(\text{/EN/US}, x)$, is a function (in this case, the product) of the individual predicate instance probabilities; thus, if one predicate returns a wider range of probabilities, it will be more influential in the final ranking.

The query ranking objective encourages good rankings for entire queries instead of single predicates. The data for this objective consists of tuples, $\{(\ell_i, e_i)\}_{i=1}^{n}$, of a query ℓ_i with an entity answer e_i (Figure 5.3, bottom right). Each ℓ_i is a function with exactly one entity argument, and $\ell_i(e)$ is a conjunction of predicate instances. For example, the last query in Figure 5.3 is a function of one argument z, and $\ell(e)$ is a single predicate instance, $\textsc{'s}(e, \textsc{/m/072qmc})$. This objective aims to rank the observed entity answer above unobserved entities for each query:

$$O_Q(\theta, \phi) \;=\; \sum_{i=1}^{n} \log P_{rank}(\ell_i, e_i, e_i')$$

P_{rank} generalizes the approximate ranking probability defined by the predicate ranking objective to more general queries. The expression $\sigma(\theta_c^T(\phi_e - \phi_{e'}))$ in the predicate ranking objective can be viewed as an approximation of the probability that e is ranked above e' in category c. P_{rank} uses this approximation for each individual predicate in the query. For example, given the query $\ell = \lambda x.c(x) \wedge r(x, y)$ and entities (e, e'), $P_{rank}(\ell, e, e') = \sigma(\theta_c(\phi_e - \phi_{e'})) \times \sigma(\theta_r(\phi_{(e,y)} - \phi_{(e',y)}))$. This notion generalizes to other logical connectives in the same fashion as probabilistic database queries. During training, e_i' is resampled on each pass over the data set from the empirical distribution of entities such that (ℓ_i, e_i') does not occur in the training data.

When ℓ's body consists of a conjunction of predicates, the query ranking objective simplifies considerably. In this case, ℓ can be described as three sets of one-place predicates: categories $C(\ell) = \{\lambda x.c(x)\}$, left arguments of relations $R_L(\ell) = \{\lambda x.r(x, y)\}$, and right arguments of relations $R_R(\ell) = \{\lambda x.r(y, x)\}$. Furthermore, P_{rank} is a product so we can distribute the log:

$$\begin{aligned} O_Q(\theta, \phi) \;=\; \sum_{i=1}^{n} & \sum_{\lambda x.c(x) \in C(\ell_i)} \log \sigma(\theta_c(\phi_{e_i} - \phi_{e_i'})) \\ & + \sum_{\lambda x.r(x,y) \in R_L(\ell_i)} \log \sigma(\theta_r(\phi_{(e_i,y)} - \phi_{(e_i',y)})) \\ & + \sum_{\lambda x.r(y,x) \in R_R(\ell_i)} \log \sigma(\theta_r(\phi_{(y,e_i)} - \phi_{(y,e_i')})) \end{aligned}$$

This simplification reveals that the main difference between O_Q and O_P is the sampling of the unobserved entities and tuples. O_P samples them in an unconstrained fashion from their empirical distributions for every predicate. O_Q considers the larger context in which each predicate occurs, with two major effects. First, more negative examples are generated for categories because the logical forms ℓ are more specific. For example, both "president of Sprint" and "president of the US" generate queries containing the PRESIDENT predicate; O_Q will use entities that only occur with one of these queries as negative examples for the other. Second, the relation parameters are trained to rank tuples with a shared argument, as opposed tuples in general.

5.6 Evaluation

The evaluation uses a question answering task to evaluate the performance of this chapter's open predicate vocabulary approach to compositional semantics. Each test example is a (compositional) natural language question whose answer is a set of Freebase entities. The presented approach is compared to several baselines based on prior work, as well as to a manually constructed Freebase query for each example.

5.6.1 Data

The training and test data for the evaluation are derived from the Clueweb09 web corpus[3] with the corresponding Google FACC entity linking (Gabrilovich et al., 2013). The training data is derived from 3 million webpages and contains 1.7m predicate instances, 1.1m queries, 150k entities and 130k entity pairs. Predicates that appeared fewer than 6 times in the training data were replaced with the predicate UNK, resulting in 16k categories and 1.4k relations.

The test data consists of fill-in-the-blank natural language questions such as "Incan emperor __" or "Cunningham directed Auchtre's second music video __." These questions were created by applying the training data generation process (Section 5.5.1) to a collection of held-out web-

[3] http://www.lemurproject.org/clueweb09.php

pages. Each natural language question has a logical form containing at least one category and one relation.

These experiments do not use existing data sets for semantic parsing into Freebase, as the goal is to model the semantics of language that cannot necessarily be modelled using the Freebase schema. Existing data sets, such as Free917 (Cai and Yates, 2013b) and WebQuestions (Berant et al., 2013), would not evaluate performance on this subset of language. Consequently, the evaluation uses a new data set with unconstrained language. However, to provide a comparison against approaches that semantically parse to Freebase, the experiments compare against annotated Freebase queries (Section 5.6.5).

All of the data for this evaluation is available online at `http://rtw.ml.cmu.edu/tacl2015_csf`.

5.6.2 Methodology

The evaluation methodology is inspired by information retrieval evaluations (Manning et al., 2008). Each system predicts a ranked list of 100 answers for each test question. The top 30 answers of each system are then pooled and manually judged for correctness. The correct answers from the pool are then used to evaluate the precision and recall of each system. In particular, the evaluation computes average precision (AP) for each question and reports the mean average precision (MAP) across all questions. It also reports a weighted version of MAP, where each question's AP is weighted by its number of correct answers. Average precision is computed as $\frac{1}{m}\sum_{k=1}^{m} \mathrm{Prec}(k) \times \mathrm{Correct}(k)$, where $\mathrm{Prec}(k)$ is the precision at rank k, $\mathrm{Correct}(k)$ is an indicator function for whether the kth answer is correct, and m is the number of returned answers (at most 100).

Statistics of the annotated test set are shown in Table 5.2. A consequence of the unconstrained data generation approach is that some test queries are difficult to answer: of the 220 queries, at least one system is able to produce a correct answer for 116. The remaining queries are mostly

# of queries	220
Avg. # of predicates / query	2.77
Avg. # of categories / query	1.75
Avg. # of relations / query	1.02
Avg. # of answers / query	1.92
# of queries with >1 answer	116

Table 5.2: Statistics of the test data set.

unanswerable because they reference rare entities unseen in the training data.

5.6.3 Models and Baselines

The evaluation includes two baseline models based on existing techniques. The CORPUSLOOKUP baseline answers test queries by directly using the predicate instances in the training data as its knowledge base. For example, given the query $\lambda x.\textsc{ceo}(x) \wedge \textsc{of}(x, \textsc{/en/sprint})$, this model will return the set of entities e such that $\textsc{ceo}(e)$ and $\textsc{of}(e, \textsc{/en/sprint})$ both appear in the training data. All answers found in this fashion are assigned probability 1.

The CLUSTERING baseline first clusters the predicates in the training corpus, then answers queries using the clustered predicates. The clustering aggregates predicates with similar denotations, ideally identifying synonyms to smooth over sparsity in the training data. This baseline is closely based on Lewis and Steedman (Lewis and Steedman, 2013), though is also conceptually related to approaches such as DIRT (Lin and Pantel, 2001) and USP (Poon and Domingos, 2009; Titov and Klementiev, 2011). Clustering is performed using the Chinese Whispers algorithm (Biemann, 2006) with predicate similarity scores calculated as the cosine similarity of their TF-IDF weighted entity count vectors. The denotation of each cluster is the union of the denotations of the clustered predicates, and each entity in the denotation is assigned probability 1.

The evaluation also includes two probabilistic database models, FACTORIZATION (O_P) and FACTORIZATION (O_Q), trained using the two objective functions described in Sections 5.5.2 and 5.5.3, respectively. Both objectives were optimized by performing 100 passes over the training

	MAP	Weighted MAP
CLUSTERING	0.224	0.266
CORPUSLOOKUP	0.246	0.296
FACTORIZATION (O_P)	0.299	0.473
FACTORIZATION (O_Q)	0.309	0.492
ENSEMBLE (O_P)	0.391	0.614
ENSEMBLE (O_Q)	0.406	0.645
Upper bound	0.527	1.0

Table 5.3: Mean average precision for our question answering task. The difference in MAP between each pair of adjacent models is statistically significant ($p < .05$) via the sign test.

data with AdaGrad (Duchi et al., 2011) using an L2 regularization parameter of $\lambda = 10^{-4}$. The predicate and entity embeddings have 300 dimensions.

Finally, initial experiments found that CORPUSLOOKUP has high precision but low recall, while both matrix factorization models have high recall with somewhat lower precision. This observation suggested that an ensemble of CORPUSLOOKUP and FACTORIZATION could outperform either model individually. The evaluation includes two ensembles, ENSEMBLE (O_P) and ENSEMBLE (O_Q), created by calculating the probability of each predicate as a 50/50 mixture of each model's predicted probability.

5.6.4 Results

Table 5.3 shows the results of the MAP evaluation, and Figure 5.4 shows a precision/recall curve for each model. The MAP numbers are somewhat low because almost half of the test queries have no correct answers and all models get an average precision of 0 on these queries. The upper bound on MAP is the fraction of queries with at least 1 correct answer. Note that the models perform well on the answerable queries, as reflected by the ratio of the achieved MAP to the upper bound. The weighted MAP metric also corrects for these unanswerable queries, as they are assigned 0 weight in the weighted average.

These results demonstrate several findings. First, both FACTORIZATION models outperform

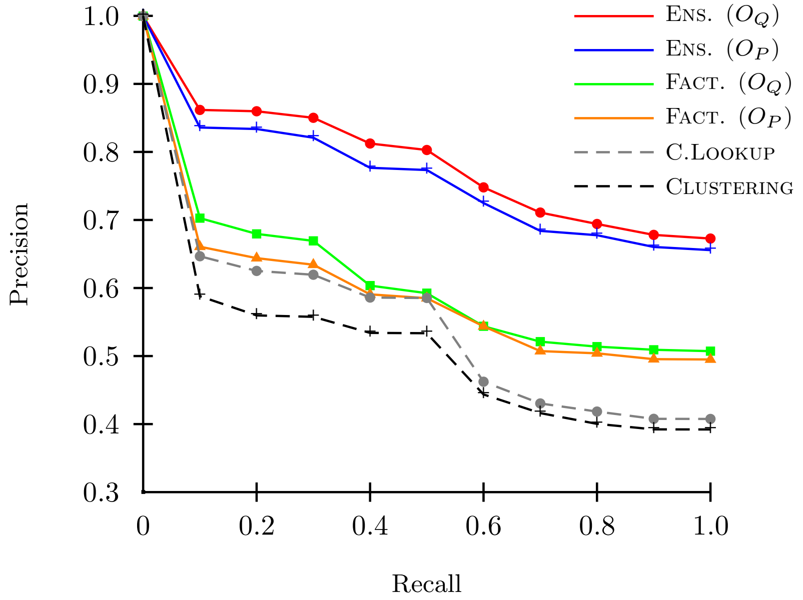


Figure 5.4: Averaged 11-point precision/recall curves for the 116 answerable test questions.

the baselines in both MAP and weighted MAP. The performance improvement seems to be most significant in the high recall regime (right side of Figure 5.4). Second, the query ranking objective O_Q improves performance over the predicate ranking objective O_P by 2-4% on the answerable queries. The precision/recall curves show that this improvement is concentrated in the low recall regime. Finally, the ensemble models are considerably better than their component models; however, even in the ensembled models, O_Q outperforms O_P by a few percent.

5.6.5 Comparison to Semantic Parsing to Freebase

A natural question is whether this chapter's open vocabulary approach outperforms a closed vocabulary approach for the same problem, such as semantic parsing to Freebase. In order to answer this question, this evaluation compares the best performing model from the previous evaluation to a manually constructed Freebase query for each test example. This comparison illuminates the relative advantages of open and closed predicate vocabularies.

A Freebase MQL query was annotated for each natural language question in the test data set. This annotation is somewhat subjective, as many of the questions can only be inexactly mapped

Question: the Midwest supermarket chain ____
MQL Query:

```
[{"/business_operation/industry": {"mid": "/en/supermarket"},
  "/organization/organization/geographic_scope":
    [{"mid": "/en/midwest"}],
  "mid": null,
  "name": null}]
```

Question: ____ mayor Stephen R. Reed
MQL Query:

```
[{"/government/politician/government_positions_held":
  [{"/government/government_position_held/basic_title": {"mid": "/en/mayor"},
    "/government/government_position_held/jurisdiction_of_office":
      {"mid": null, "name": null}}],
  "mid": "/en/stephen_r_reed"}]
```

Figure 5.5: Example questions from the test data set and their annotated Freebase MQL queries. Entity IDs in the queries have been replaced for readability. Running these queries against Freebase returns a list of values for the fields whose value is `null`.

# of annotated queries	142
>1 answer	95
no answers	47
# unannotatable	78

Table 5.4: Statistics of the Freebase MQL queries annotated for the test data set.

on to the Freebase predicate vocabulary. The following guidelines were used in performing the mapping: (1) all relations in the text must be mapped to one or more Freebase relations, (2) all entities mentioned in the text must be included in the query, (3) adjective modifiers can be ignored and (4) entities not mentioned in the text may be included in the query. The fourth condition is necessary because many one-place predicates, such as MAYOR(x), are represented in Freebase using a binary relation to a particular entity, such as GOVERNMENT_OFFICE/TITLE(x, /EN/MAYOR). Some example annotated Freebase MQL queries for the test data are shown in Figure 5.5.

Statistics of the annotated queries are shown in Table 5.4. Coverage is reasonably high: Freebase queries were annotated for 142 examples (65% of the test set). The remaining unannotatable examples are due to missing predicates in Freebase, such as a relation defining the emperor of the Incan empire. Of the 142 annotated Freebase queries, 95 of them return at least one entity answer. The queries with no answers typically reference uncommon entities which have few or no known relation instances in Freebase. The annotated queries contain 2.62 Freebase relations on average.

This evaluation compares ENSEMBLE (O_Q) to the manually annotated Freebase queries using the same pooled evaluation methodology. The set of correct answers contains the correct predictions of ENSEMBLE (O_Q) from the previous evaluation along with all answers returned by Freebase. (That is, Freebase is assumed to have perfect precision.)

Results from this evaluation are shown in Table 5.5. In terms of overall MAP, Freebase outperforms ENSEMBLE (O_Q) by a fair margin. However, this initial impression belies a more complex reality, which is shown by Table 5.6. This table compares both approaches by their relative performance on each test example. On approximately one-third of the examples, Freebase has a higher AP than ENSEMBLE (O_Q). On another third, ENSEMBLE (O_Q) has a higher AP than Freebase. On the final third, both approaches perform equally well – these are typically examples where neither approach returns any correct answers (67 of the 75). Freebase outperforms in the overall MAP evaluation because it tends to return more correct answers to each question.

Note that the annotated Freebase queries have several advantages in this evaluation. First, Freebase contains significantly more predicate instances than the training data, which allows it to produce more complete answers. Second, the Freebase queries correspond to the performance of a perfect semantic parser, while current semantic parsers achieve accuracies around 68% (Berant and Liang, 2014).

The results from this experiment suggest that closed and open predicate vocabularies are complementary. Freebase produces high quality answers when it covers a question. However, many of the remaining questions can be answered correctly using an open vocabulary approach. This evaluation also suggests that recall is a limiting factor of the presented approach; however, recall can be improved by simply using a larger training corpus.

	MAP
ENSEMBLE (O_Q)	0.263
Freebase	0.385

Table 5.5: Mean average precision of the best performing model from the previous evaluation compared to a manually constructed Freebase query for each test question. The numbers in this table are not comparable to the numbers in Table 5.3 as the correct answers for each question are different.

	# of queries
Freebase higher AP	75 (34%)
equal AP	75 (34%)
ENSEMBLE (O_Q) higher AP	70 (31%)

Table 5.6: Question-by-question comparison of model performance. Each test question is placed into one of the three buckets above, depending on whether Freebase or ENSEMBLE (O_Q) achieved a better average precision (AP) for the question.

5.7 Discussion

This chapter presents an approach to building a language understanding system with an open predicate vocabulary. The approach defines a probabilistic model over denotations (sets of Freebase entities) conditioned on an input text. The model has two components: a rule-based semantic parser that produces a logical form for the text, and a learned probabilistic database representing a distribution over possible knowledge bases. Inference in the model is performed by computing the marginal probability, over all possible knowledge bases, that an entity is an element of the text's denotation. A training phase learns the probabilistic database by applying probabilistic matrix factorization to query/answer pairs that are automatically derived by running the rule-based semantic parser on a large, entity-linked web corpus. Unlike in previous chapters, a knowledge base is not required to train the system nor to compute denotations. An experimental analysis demonstrates that this approach outperforms several baselines, and can answer many questions that cannot be answered by semantic parsing into Freebase.

Scaling up the size of the training corpus is a direction for future work. Both stages of training, data generation and matrix factorization, can be parallelized using a cluster. It should be feasible to increase the quantity of training data by a factor of 10-100, for example, to train on all of ClueWeb. Another possibility is to use a universal schema and directly include predicate instances from Freebase in the matrix factorization (Riedel et al., 2013). Increasing the size of the training corpus should improve recall, especially for uncommon entities.

There is an interesting connection between this work and compositional semantics with vector space models (Krishnamurthy and Mitchell, 2013; Socher et al., 2011). In both cases, the meanings of individual words are represented as vectors. In that previous work, semantic composition is performed directly in the space of vectors (or matrices, etc.). However, in this chapter, these vectors are first mapped to sets of entities, and composition is performed using using logical operations on these sets. This approach has the advantage of being able to straightforwardly model phenomena that are challenging for vector space models, such as the action of subordinating conjunctions (e.g., “that” in “car that I drive”).

The positioning of this chapter according to the axes of variation is as follows:

1. **Learning the knowledge base** – The knowledge base is learned using probabilistic matrix factorization.
2. **Predicate vocabulary** – An open predicate vocabulary is derived from an entity-linked text corpus based on the output of a rule-based semantic analysis system.
3. **Semantic parser supervision** – Query/answer pairs, derived from entity-linked text, are used to train the probabilistic database.
4. **Syntactic information** – A syntactic CCG parser, trained on CCGbank, is used to perform the rule-based semantic analysis. The mapping from syntax to semantics is unlearned.

The major difference between this chapter and the previous one is the open predicate vocabulary. This difference influences other aspects of the system’s design, with several consequences. First, it forces the knowledge base to be automatically learned, as no predicate instances are given as input. Second, it enables the system to use a fixed mapping from syntax to semantics, as it is no longer necessary to map between words and symbols in a knowledge base. The overall approach is, in some sense, the exact opposite of the semantic parser from the previous chapter: there, the semantic parser is learned and the knowledge base is fixed; here, the knowledge base is learned and the semantic parser is fixed. From a machine learning perspective, training the knowledge base via matrix factorization is considerably easier than training the semantic parser with distant supervision, as there is no difficult inference problem during training.

The evaluation demonstrated that the open predicate vocabulary enables the system to answer many questions that cannot be answered by Freebase. However, Freebase retained a recall advantage when it could answer a question. These results suggest that there are complementary roles for both open and closed predicate vocabularies. A direction for future work is combining these approaches, perhaps by constructing a semantic parser capable of using both kinds of predicates.

A natural question is whether the rule-based semantic analysis system could be replaced with a learned semantic parser. Although syntactic parsers for English can be trained with existing treebanks, these are expensive to produce and are not available for many other languages. Furthermore, the syntactic parses in the treebank may differ from the ideal syntactic parse for the purpose of semantic analysis. For example, type-raising is often used to perform quantifier scoping, but the annotations in CCGbank do not relate these two phenomena. The next chapter presents an algorithm that jointly learns both a knowledge base *and* a semantic parser.

Chapter 6

Semantic Parsing for Grounded Environments

Abstract

This chapter presents Logical Semantics with Perception (LSP), a model for grounded language acquisition that learns to map natural language statements to their referents in a physical environment. For example, given an image, LSP can map the statement “blue mug on the table” to the set of image segments depicting blue mugs on tables. LSP has two components: a semantic parser that is used to interpret natural language statements, and a perception component that produces knowledge bases from physical environments. This chapter also presents a weakly supervised training algorithm for LSP that estimates the parameters of these components using annotated referents for entire statements, without referents of individual words or the parse structure of the statement. Experiments on scene understanding and geographical question answering demonstrate that LSP outperforms existing models and that weakly supervised training is competitive with fully supervised training while requiring significantly less annotation effort.

Learning the mapping from natural language to physical environments is a central problem for natural language semantics. From a practical perspective, there are many important applications for this technology, such as enabling natural language interactions with robots and other embodied systems. For example, a robot given the command "get the blue mug on the table" must be able to identify the "blue mug on the table" in the environment in order to correctly interpret the command. From a theoretical perspective, model-theoretic semantics argues that understanding natural language is equivalent to understanding the real world situations in which statements are true or false.

The problem of learning to map from natural language expressions to their referents in an environment is known as *grounded language acquisition*. Although there are various settings for this problem, the most interesting setting is when environments consist of raw sensor data – for example, an environment could be an image collected from a robot's camera. In such applications, grounded language acquisition has two subproblems: *semantic parsing*, learning the compositional structure of natural language; and *perception*, learning the environmental referents of individual words.

This chapter introduces Logical Semantics with Perception (LSP), a model for grounded language acquisition that jointly learns to semantically parse language and perceive the world. LSP models a mapping from natural language queries to sets of objects in a real-world environment. The input to LSP is an environment containing objects, such as a segmented image (Figure 6.1a), and a natural language query, such as "the things to the right of the blue mug." Given these inputs, LSP produces (1) a logical knowledge base describing objects and relationships in the environment and (2) a semantic parse of the query. Evaluating the semantic parse on the predicted knowledge base gives the denotation (i.e., real-world referents) of the natural language query (Figure 6.1b).[1]

[1] This chapter treats declarative sentences as if they were queries about their subject, e.g., the denotation of "the mug is on the table" is the set of mugs on tables. Typically, the denotation of a sentence is either true or false; this treatment is strictly more general, as a sentence's denotation is nonempty if and only if the sentence is true.

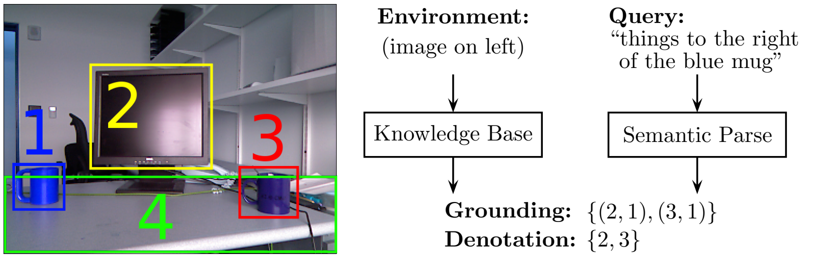
This chapter further presents a weakly supervised training algorithm that estimates parameters for LSP using natural language queries annotated with their denotations in an environment (Figure 6.1c). Such denotations represent a natural form of training data that could be obtained by, for example, following a user's pointing gesture while they describe an object (Kollar et al., 2013). Training from denotations significantly reduces the amount of manual annotation required. In comparison, a fully supervised approach requires both a corpus of natural language queries annotated with logical forms and a corpus of environments annotated with knowledge bases. These annotations are not obtainable through natural interactions with non-expert users.

Experiments on two different applications demonstrate that (1) LSP outperforms the most comparable state-of-the-art approach to grounded language acquisition and (2) weakly supervised training achieves nearly the same accuracy as fully supervised training. The first application is scene understanding, where LSP maps descriptions of objects to image segments. The second application is geographical question answering, where LSP answers questions about locations given a map of their boundaries represented as latitude/longitude polygons. LSP correctly answers 34% more questions in geographical question answering and 16% more questions in scene understanding than a comparable state-of-the-art model (Matuszek et al., 2012). Furthermore, in both applications, weakly supervised training achieves an accuracy within 6% of that achieved by fully supervised training, while requiring significantly less annotation effort.

The contents of this chapter are derived from (Krishnamurthy and Kollar, 2013).

6.1 Logical Semantics with Perception

Logical Semantics with Perception (LSP) is a model for grounded language acquisition. LSP accepts as input a natural language statement and an environment and outputs the objects in the environment denoted by the statement. The LSP model has three components: perception, semantic parsing and evaluation (see Figure 6.2). The idea behind this factorization is simple: given a knowledge base describing the environment, a semantic parser can map natural language



Language	Denotation
The mugs	{1, 3}
The objects on the table	{1, 2, 3}
There is an LCD monitor	{2}
Is the blue mug right of the monitor?	{}
The monitor is behind the blue cup.	{2}

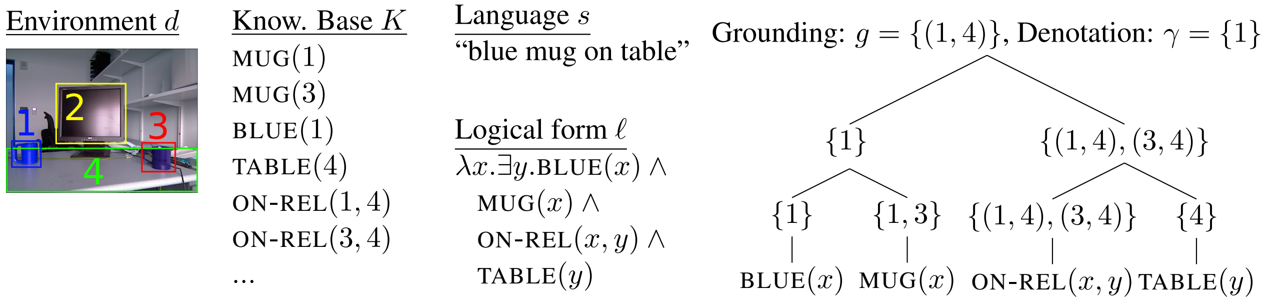
(a) An environment containing 4 objects. (b) LSP predicting the environmental referents of a natural language query. (c) Training examples for weakly supervised training.

Figure 6.1: LSP applied to scene understanding. Given an environment containing a set of objects (left), and a natural language query, LSP produces a semantic parse, logical knowledge base, grounding and denotation (middle), using only language/denotation pairs (right) for training.

statements to sets of objects in the environment. Since no knowledge base is provided as input, the model includes a perception component that predicts a knowledge base from an environment. Given this knowledge base, the standard semantic parsing and evaluation pipeline is used to predict denotations.

The output of LSP can be either a *denotation* or a *grounding*. A denotation is the usual output of a semantic parser: the set of entity referents for the query as a whole. For the purpose of evaluation, it is useful to also output a grounding, which is the set of entity referents for each noun phrase in the query. The distinction between these two outputs is shown in Figure 6.1b. In this example, the denotation is the set of "things to the right of the blue mug," which does not include the blue mug itself. On the other hand, the grounding is a set of tuples, where the first component is the referent of "things" and the second component the referent of "blue mug." Note that the denotation is the projection of the grounding's tuples on to their first element. Groundings are used in the evaluation to determine if the model truly identified all of the named objects in a query. However, only denotations are used during training, so groundings are ignored in the following model description.

Formally, LSP is a linear model f that predicts a denotation γ given a natural language statement s in an environment d. As shown in Figure 6.3, the structure of LSP factors into



(a) Perception f_{per} produces a knowledge base K from an environment d using per-predicate classifiers.

(b) Semantic parsing f_{prs} maps language s to a logical form ℓ.

(c) Evaluation f_{eval} evaluates a logical form ℓ on a logical knowledge base K to produce a grounding g and denotation γ.

Figure 6.2: Overview of Logical Semantics with Perception (LSP).

perception (f_{per}), semantic parsing (f_{prs}) and evaluation (f_{eval}) components using several latent variables:

$$f(\gamma, K, \ell, t, s, d; \theta) = f_{per}(K, d; \theta_{per}) + f_{prs}(\ell, t, s; \theta_{prs}) + f_{eval}(\gamma, K, \ell)$$

An input to LSP is a set of one- and two-argument predicates (i.e., categories C and relations R). These predicates are the interface between LSP's perception and parsing components. The perception function f_{per} takes an environment d and produces a knowledge base K that assigns truth values to instances of these predicates using parameters θ_{per}. The semantic parser f_{prs} takes a natural language statement s and produces a logical form ℓ and syntactic parse t using parameters θ_{prs}. Finally, the evaluation function f_{eval} deterministically evaluates the logical form ℓ on the knowledge base K to produce a denotation γ. Example inputs and outputs of these components are illustrated in Figure 6.2.

The following sections describe the perception function (Section 6.1.1), semantic parser (Section 6.1.2), evaluation function (Section 6.1.3), and inference (Section 6.1.4) in more detail.

6.1.1 Perception function

The perception function f_{per} constructs a logical knowledge base K given an environment d. Each environment contains a collection of entities $e \in E_d$. Perception produces a knowledge

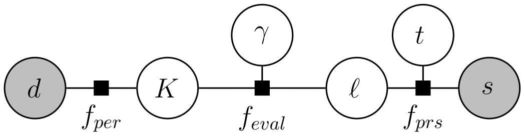


Figure 6.3: Factor graph of LSP. The environment d and language s are given as input, from which the model predicts a logical knowledge base K, logical form ℓ, syntactic tree t and denotation γ.

base consisting of ground predicate instances over these entities. For example, in Figure 6.2a, the entire image is the environment, and each image segment is an entity. The logical knowledge base K contains the shown predicate instances, where the categories include BLUE, MUG and TABLE, and the relations include ON-REL.

The perception function scores logical knowledge bases using a collection of per-predicate binary classifiers. These classifiers independently assign a score to whether each entity (entity pair) is an element of each category (relation). Let $\gamma^c \in K$ denote the set of entities which are elements of category c; similarly, let $\gamma^r \in K$ denote the set of entity pairs which are elements of the relation r. Given these sets, the score of a logical knowledge base K factors into per-relation and per-category scores h:

$$f_{per}(K, d; \theta_{per}) = \sum_{c \in C} h(\gamma^c, d; \theta^c_{per}) + \sum_{r \in R} h(\gamma^r, d; \theta^r_{per})$$

The per-predicate scores are in turn given by a sum of per-element classification scores:

$$\begin{aligned} h(\gamma^c, d; \theta^c_{per}) &= \sum_{e \in E_d} \gamma^c(e)(\theta^c_{per})^T \phi_{cat}(e) \\ h(\gamma^r, d; \theta^r_{per}) &= \sum_{(e_1, e_2) \in E_d} \gamma^r(e_1, e_2)(\theta^r_{per})^T \phi_{rel}(e_1, e_2) \end{aligned}$$

Each term in the above sums represents a single binary classification, calculating a score for whether a single entity (entity pair) belongs to a particular category (relation). γ^c and γ^r are indicator functions for the sets they denote, i.e., $\gamma^c(e) = 1$ for entities e in the set, and 0 otherwise. Similarly, $\gamma^r(e_1, e_2) = 1$ for entity pairs (e_1, e_2) in the set, and 0 otherwise. The features of these classifiers are given by ϕ_{cat} and ϕ_{rel}, which are functions that map entities and entity pairs to feature vectors. (Section 6.3.2 describes the features used for each application in the evaluation.) The parameters of these classifiers are given by θ^c_{per} and θ^r_{per}. The perception parameters θ_{per} contain one such set of parameters for every category and relation, i.e., $\theta_{per} = \{\theta^c_{per} : c \in C\} \cup \{\theta^r_{per} : r \in R\}$.

6.1.2 Semantic parser

The semantic parser maps a natural language statement into a database query against the perception function's predicted knowledge base K. This component is a CCG semantic parser with a heuristically constructed lexicon and grammar. The lexicon includes entries such as:

$$\begin{array}{rcl}
\text{mug} & := & N : \lambda x.\textsc{mug}(x) \\
\text{mugs} & := & N : \lambda x.\textsc{mug}(x) \\
\text{on} & := & (N\backslash N)/N : \lambda f.\lambda g.\lambda x.\exists y.f(y) \wedge g(x) \wedge \textsc{on-rel}(x,y) \\
\text{on} & := & PP/N : \lambda f.f \\
\text{placed} & := & (S\backslash N)/PP : \lambda f.\lambda g.\lambda x.\exists y.f(y) \wedge g(x) \wedge \textsc{place-rel}(x,y) \\
\text{the} & := & N/N : \lambda f.f \\
\text{blue} & := & N/N : \lambda f.\lambda x.f(x) \wedge \textsc{blue}(x)
\end{array}$$

Generally, this grammar uses categories to represent the meanings of nouns and adjectives, and relations to represent verbs and prepositions. The grammar does not model comparatives or other phenomana that require higher-order functions. Section 6.3.3 describes the part-of-speech heuristics that are used to produce the lexicon.

The semantic parser is a linear model over semantic parses (ℓ, t) given language s:

$$f_{prs}(\ell, t, s; \theta_{prs}) = \theta^T_{prs}\phi_{prs}(\ell, t, s)$$

$\phi_{prs}(\ell, t, s)$ represents a feature function mapping semantic parses to vectors of feature values. In this chapter, ϕ_{prs} contains lexicon indicator features and combinator features (see Section 2.2.3).

6.1.3 Evaluation function

The evaluation function takes a logical form ℓ and a knowledge base K and produces a denotation γ. This function is unlearned and deterministic: the denotation γ is always the result of evaluating ℓ as a query on K. The evaluation function assigns score 0 to this denotation, and score $-\infty$ to all other denotations.

6.1.4 Inference

The test-time inference problem in LSP is to predict a denotation γ for language s in an environment d. This inference problem is straightforward due to the deterministic structure of f_{eval}. The highest-scoring γ can be found by independently maximizing f_{prs} and f_{per} to find the highest-scoring logical form ℓ and knowledge base K. The highest-scoring denotation is then produced by evaluating ℓ on K.

Another inference problem occurs during training: identify the highest-scoring logical form and knowledge base that produce a particular denotation. This problem is no longer straightforward, since observing γ couples ℓ and K. An approximate inference algorithm for this problem is described in Section 6.2.2.

6.2 Training from Denotations

This section describes a weakly supervised training algorithm for LSP that estimates parameters using a corpus of natural language queries with annotated denotations. The algorithm jointly trains both the parsing and the perception components of LSP to best predict the denotations of the annotated training queries. Intuitively, this procedure trains a semantic parser with an *unknown* knowledge base, simultaneously learning how to semantically parse language and populate the knowledge base.

The weakly supervised algorithm trains LSP as a maximum margin Markov network (Taskar et al., 2004), a structured analog of a support vector machine, using the stochastic subgradient method. The main difficulty is computing the subgradient, which requires computing values for the model's hidden variables (i.e., the logical knowledge base K and semantic parse ℓ) that produce the annotated denotations.

6.2.1 Stochastic Subgradient Method

The training data for the weakly supervised algorithm is a collection $\{(s^i, \gamma^i, d^i)\}_{i=1}^n$ consisting of language s^i paired with its denotation γ^i in environment d^i. Given this data, LSP's parameters $\theta = [\theta_{prs}, \theta_{per}]$ are estimated by minimizing the following objective function:

$$O(\theta) = \frac{\lambda}{2}||\theta||^2 + \frac{1}{n}\left[\sum_{i=1}^{n} \zeta_i\right] \tag{6.1}$$

where λ is a regularization parameter that controls the trade-off between model complexity and slack penalties. The slack variable ζ_i represents a margin violation penalty for the ith training example, defined as:

$$\zeta_i = \max_{\gamma,K,\ell,t}\left[f(\gamma, K, \ell, t, s^i, d^i; \theta) + \text{cost}(\gamma, \gamma^i)\right] - \max_{K,\ell,t} f(\gamma^i, K, \ell, t, s^i, d^i; \theta)$$

The above expression is a structured analog of the hinge loss, where $\text{cost}(\gamma, \gamma^i)$ is the margin by which γ^i's score must exceed γ's score. Here, $\text{cost}(\gamma, \gamma^i)$ is the Hamming cost: it adds a cost of 1 for each entity e such that $\gamma^i(e) \neq \gamma(e)$.

This objective is optimized using the stochastic subgradient method (Ratliff et al., 2006). The first step in computing the subgradient g^i is to solve the following inference problems:

$$\hat{\gamma}, \hat{K}, \hat{\ell}, \hat{t} \leftarrow \arg\max_{\gamma,K,\ell,t} f(\gamma, K, \ell, t, s^i, d^i; \theta^j) + \text{cost}(\gamma, \gamma^i) \tag{6.2}$$

$$K^*, \ell^*, t^* \leftarrow \arg\max_{K,\ell,t} f(\gamma^i, K, \ell, t, s^i, d^i; \theta^j) \tag{6.3}$$

The first set of values (e.g., $\hat{\ell}$) are the best explanation for the denotation $\hat{\gamma}$ which most violates the margin constraint. The second set of values (e.g., ℓ^*) are the best explanation for the true denotation γ^i. The subgradient update increases the weights of features that explain the true denotation while decreasing the weights of features that explain the denotation violating the margin. The subgradient factors into parsing and perception components: $g^i = [g^i_{prs}, g^i_{per}]$. The parsing subgradient is:

$$g^i_{prs} = \phi_{prs}(\hat{\ell}, \hat{t}, s^i) - \ \phi_{prs}(\ell^*, t^*, s^i)$$

The subgradient of the perception parameters θ_{per} factors into subgradients of the category and relation classifier parameters. Recall that $\theta_{per} = \{\theta^c_{per} : c \in C\} \cup \{\theta^r_{per} : r \in R\}$. Let $\hat{\gamma}^c \in \hat{K}$ be the best margin-violating set of entities for c, and $\gamma^{c*} \in K^*$ be the best truth-explaining set of entities. Similarly define $\hat{\gamma}^r$ and γ^{r*}. The subgradients of the category and relation classifier parameters are:

$$g^{i,c}_{per} = \sum_{e \in E_{d^i}} (\hat{\gamma}^c(e) - \gamma^{c*}(e))\, \phi_{cat}(e)$$

$$g^{i,r}_{per} = \sum_{(e_1,e_2) \in E_{d^i}} (\hat{\gamma}^r(e_1, e_2) - \gamma^{r*}(e_1, e_2))\, \phi_{rel}(e_1, e_2)$$

For each predicate, these subgradients correspond to the sum of several SVM-style subgradients, one per entity or entity pair in the environment.

6.2.2 Inference: Computing the Subgradient

Solving the maximizations in Equations 6.2 and 6.3 is challenging because the weights placed on the denotation γ (either by $\text{cost}(\gamma, \gamma^i)$ or conditioning $\gamma = \gamma^i$) couple f_{prs} and f_{per}. This coupling makes inference in LSP no longer tractable. Exactly solving these inference problems requires enumerating all possible logical forms ℓ, and then, for each logical form, finding the highest-scoring knowledge base K by propagating the weights on γ back through f_{eval}.

This exact inference procedure suggests a two-step approximate inference algorithm. The first step is to perform a beam search over CCG parses to produce k possible logical forms

$\ell_1, ..., \ell_k$. The second step is to find the best knowledge base K given each logical form ℓ_i; this problem can be formulated as an integer linear program (ILP). The highest-scoring logical form/knowledge base pair is the approximate maximizer. In the experiments, the first step uses a beam size of 1000, and the second step solves the ILP for each of the 10 highest-scoring logical forms.

Given a logical form ℓ output by beam search, the second step of approximate inference is to compute the best values for the knowledge base K and denotation γ:

$$\arg\max_{K,\gamma} f_{per}(K, d; \theta_{per}) + f_{eval}(\gamma, \ell, K) + \psi(\gamma) \tag{6.4}$$

Above, $\psi(\gamma)$ represents a set of weights on the entities in the predicted denotation γ. In Equation 6.2, ψ represents $\text{cost}(\gamma, \gamma^i)$, which is encoded by setting $\psi(\gamma) = \sum_{e \in E_d} \mathbf{1}[\gamma^i(e) = 0]\gamma(e) + \mathbf{1}[\gamma^i(e) = 1](1 - \gamma(e))$, where $\mathbf{1}[\cdot]$ is the indicator function. In Equation 6.3, ψ is a hard constraint enforcing $\gamma = \gamma^i$, which is encoded by setting $\psi(\gamma) = -\infty \times \sum_{e \in E_d} \mathbf{1}[\gamma^i(e) = 0]\gamma(e) + \mathbf{1}[\gamma^i(e) = 1](1 - \gamma(e))$, with the convention that $\infty \times 0 = 0$.

The maximization in Equation 6.4 can be written as an ILP. For each category c, relation r and entity $e_1, e_2 \in E_d$, the ILP includes binary variables $\gamma^c(e_1)$ and $\gamma^r(e_1, e_2)$ representing whether $c(e_1)$ and $r(e_1, e_2)$ are true in K. It similarly includes binary variables $\gamma(e)$ for the denotation γ. f_{per} and ψ are linear functions of these variables, and the ILP objective is:

$$f_{per}(K, d; \theta_{per}) + \psi(\gamma) = \sum_{e_1 \in E_d} \sum_{c \in C} w^c_{e_1} \gamma^c(e_1) + \sum_{e_1, e_2 \in E_d} \sum_{r \in R} w^r_{e_1,e_2} \gamma^r(e_1, e_2) + \psi(\gamma)$$

where the weights $w^c_{e_1}$ and $w^r_{e_1,e_2}$ determine how likely it is that each entity or entity pair belongs to the predicates c and r:

$$w^c_{e_1} = (\theta^c_{per})^T \phi_{cat}(e_1)$$
$$w^r_{e_1,e_2} = (\theta^r_{per})^T \phi_{rel}(e_1, e_2)$$

f_{eval} is represented in the ILP using constraints and additional auxiliary variables. These constraints couple the denotation γ and the logical knowledge base K such that γ is the result of evaluating ℓ on K. The precise structure of these constraints depends on the particular logical form.

6.3 Evaluation

The evaluation performs two major comparisons. First, it compares weakly supervised and fully supervised training to determine by how much they differ in performance. Second, it compares LSP against an ablated version of LSP similar to the model of Matuszek et al. (Matuszek et al., 2012) to provide a reference with prior work. This model cannot represent relational language, so this comparison also measures the performance impact of modelling relations. The evaluation also includes an error analysis that independently measures the performance of LSP's component semantic parser and perceptual classifiers.

The first part of this section describes some necessary set-up for the experiments. These sections describe the data sets, features, construction of the CCG lexicon, and details of the models. The data sets and additional evaluation resources are available online from `http://rtw.ml.cmu.edu/tacl2013_lsp/`.

6.3.1 Data

The evaluation evaluates LSP on two applications: scene understanding (SCENE) and geographical question answering (GEOQA). Each application's data set is a collection $\{(s^i, \gamma^i, d^i, \ell^i, K^i)\}_{i=1}^n$, consisting of natural language statements s^i with annotated denotations γ^i in environments d^i. For fully supervised training, each statement is also annotated with a logical form ℓ^i, and each environment with a knowledge base K^i. Statistics of these data sets are shown in Table 6.1 and example environments and statements are shown in Figure 6.4.

The SCENE data set consists of segmented images of indoor environments containing a number of ordinary objects such as mugs and monitors. Each image is an environment, and each

Data Set Statistics	SCENE	GEOQA
# of environments	15	10
Mean entities / environment d	4.1	6.9
Mean # of entities in denotation γ	1.5	1.2
# of statements	284	263
Mean words / statement	6.3	6.3
Mean predicates / log. form	2.6	2.8
# of preds. in annotated knowledge bases	46	38
Lexicon Statistics		
# of words in lexicon	169	288
# of lexicon entries	712	876
Mean parses / statement	15.0	8.9

Table 6.1: Statistics of the two data sets used in the evaluation and of the generated lexicons.

image segment (bounding box) is an entity. Natural language descriptions of each scene were collected from members of the Read the Web group and Amazon Mechanical Turk by asking subjects to describe the objects in each scene. Each statement was then manually annotated with a denotation and logical form, and each image with a knowledge base. In this data set, each image contains the same set of objects; note that this does not trivialize the task, as the model only observes visual features of the objects that are not consistent across environments.

The GEOQA data set consists of several maps containing entities such as cities, states, national parks, lakes and oceans. Each map is an environment, and its constituent entities are described as polygons of latitude/longitude coordinates marking their boundaries. Each entity has one known name (e.g., "Greenville") that was used to query OpenStreetMap[2] to produce its boundary polygon. In this data set, each entity occurs on average in 1.25 environments; repeated entities are mostly large locations, such as states and oceans. The language for this data set was contributed by members of the Read the Web group, who were instructed to provide a mixture of simple and complex geographical questions. Each question was then manually annotated with a denotation (its answer) and a logical form, and each map with a knowledge base.

[2] `http://www.openstreetmap.org/.`

Shared features	
bias	1
SCENE only	
Histogram of Oriented Gradients (Dalal and Triggs, 2005)	$\mathbf{R}$
mean red / green / blue pixel value	$\mathbf{R}$
std. dev. red / green / blue pixel value	$\mathbf{R}$
GEOQA only	
polygon area	$\mathbf{R}$
polygon perimeter	$\mathbf{R}$
category context indicator	$\{0,1\}$
entity indicator	$\{0,1\}$

(a) Listing of category features ϕ_{cat}

Shared features	
bias	1
polygon intersection indicator	$\{0,1\}$
intersection area	$\mathbf{R}$
AVS features (Regier and Carlson, 2001)	$\mathbf{R}$
centroid distance	$\mathbf{R}$
horizontal / vertical centroid distance	$\mathbf{R}$
left / right / above / below indicator	$\{0,1\}$
SCENE only	
(none)	
GEOQA only	
arg1 equals arg2	$\{0,1\}$
relation context indicator	$\{0,1\}$

(b) Listing of relation features ϕ_{rel}

Table 6.2: Listing of features and the range of their values used in the perception function of LSP for each application.

6.3.2 Features

Table 6.2 lists the category and relation features used by LSP's perception function in each applications. The SCENE application uses a number of visual appearance features in ϕ_{cat} to capture the visual properties of objects. The GEOQA application uses properties of each object's polygon, as well as a collection of distributional semantic features to distinguish between different types of entities (e.g., states vs. lakes). These features are derived from phrase co-occurrences with entity names in the Clueweb09 web corpus.[3] The category features ϕ_{cat} include indicators for the 20 contexts which most frequently occur with an entity's name (e.g., "X is a city"). Similar relation features are included for the 20 most frequent contexts between two entities' names (e.g., "X is the capital of Y"). Finally, both applications share a set of spatial relation features derived from the bounding polygons of each entity.

[3] http://www.lemurproject.org/clueweb09/

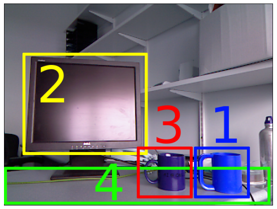
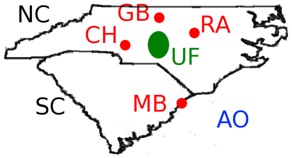
Environment d	Language s and predicted logical form ℓ	Predicted grounding	True grounding
	monitor to the left of the mugs $\lambda x.\exists y.\textsc{monitor}(x) \wedge \textsc{left-rel}(x, y) \wedge \textsc{mug}(y)$	$\{(2, 1), (2, 3)\}$	$\{(2, 1), (2, 3)\}$
	mug to the left of the other mug $\lambda x.\exists y.\textsc{mug}(x) \wedge \textsc{left-rel}(x, y) \wedge \textsc{mug}(y)$	$\{(3, 1)\}$	$\{(3, 1)\}$
	objects on the table $\lambda x.\exists y.\textsc{object}(x) \wedge \textsc{on-rel}(x, y) \wedge \textsc{table}(y)$	$\{(1, 4), (2, 4)$ $(3, 4)\}$	$\{(1, 4), (2, 4),$ $(3, 4)\}$
	two blue cups are placed near to the computer screen $\lambda x.\textsc{blue}(x) \wedge \textsc{cup}(x) \wedge \textsc{comp.}(x) \wedge \textsc{screen}(x)$	$\{(1)\}$	$\{(1, 2), (3, 2)\}$
	What cities are in North Carolina? $\lambda x.\exists y.\textsc{city}(x) \wedge \textsc{in-rel}(x, y) \wedge y = \text{NC}$	$\{(\text{CH}, \text{NC}), (\text{GB}, \text{NC})$ $(\text{RA}, \text{NC})\}$	$\{(\text{CH}, \text{NC}), (\text{GB}, \text{NC})$ $(\text{RA}, \text{NC})\}$
	What city is east of Greensboro in North Carolina? $\lambda x.\exists y, z.\textsc{city}(x) \wedge \textsc{east-rel}(x, y)$ $\wedge\, y = \text{GB} \wedge \textsc{in-rel}(y, z) \wedge z = \text{NC}$	$\{(\text{RA}, \text{GB}, \text{NC}),$ $(\text{MB}, \text{GB}, \text{NC})\}$	$\{(\text{RA}, \text{GB}, \text{NC})\}$
	What cities are on the ocean? $\lambda x.\exists y.\textsc{city}(x) \wedge \textsc{on-rel}(x, y) \wedge \textsc{ocean}(y)$	$\{(\text{CH}, \text{AO}), (\text{GB}, \text{AO}),$ $(\text{MB}, \text{AO}), (\text{RA}, \text{AO})\}$	$\{(\text{MB}, \text{AO})\}$

Figure 6.4: Example environments, statements, and model predictions from the Scene and GeoQA data sets.

6.3.3 Lexicon

The semantic parser's lexicon (Section 6.1.2) was automatically generated using part-of-speech tag heuristics. The natural language statements in each application were POS tagged using the Stanford POS tagger (Toutanova et al., 2003). The heuristics map these POS-tagged words to lexicon entries containing category and relation predicates derived from the word's lemma. Adjectives are mapped to lexicon entries containing categories. Nouns are mapped to lexicon entries containing either a category or a relation. Mapping nouns to relations is necessary for phrases such as "to the right of," where the noun "right" indicates a relation. Verbs and prepositions produce lexicon entries containing a relation. The heuristics also generate semantically-empty lexicon entries that allow words to have no physical interpretation. These rules are useful for determiners. Finally, there are special heuristics for forms of "to be" and, in GEOQA, to handle known entity names. The complete set of lexicon induction heuristics is listed in Table 6.3.

In order to facilitate an error analysis (Section 6.3.6), lexicon entries containing predicates not used in any annotated logical form were filtered out. Many of the generated predicates are intuitively meaningless (e.g., BLUE-REL(e_1, e_2)), so it is impossible to annotate ground truth instances for them. Furthermore, it is possible that training will learn a consistent interpretation for them, and that this interpretation can be used to correctly predict denotations. In this situation, it is difficult to determine the correctness of a semantic parse that uses one of these predicates. Eliminating these predicates simplifies a componentwise error analysis of the semantic parser and perception function. Statistics of the filtered lexicons are shown in Table 6.1.

6.3.4 Models and Training

The evaluation compares three models. The first model is LSP-W, which is LSP trained using the weakly supervised algorithm described in Section 6.2. The second model, LSP-CAT, replicates the model of Matuszek et al. (Matuszek et al., 2012) by restricting LSP to use only category predicates. LSP-CAT is constructed by removing all relation predicates in lexicon entries, map-

Logical Form Template	Syntactic Category	Triggering POS Tag	Example
$\lambda x.c(x)$	N	NN*, JJ*, VBN, VBG, VBD, W*	mugs := $N : \lambda x.\textsc{mug}(x)$
$\lambda f.\lambda x.c(x) \wedge f(x)$	N/N	JJ*, NN*, W	red := $N/N : \lambda f.\lambda x.\textsc{red}(x) \wedge f(x)$
$\lambda f.\lambda x.\exists y.r(x,y) \wedge f(y)$	PP_r/N	IN, TO	near := $PP_r/N : \lambda f.\lambda x.\exists y.\textsc{near-rel}(x,y) \wedge f(y)$
	N/PP	NN*, JJ*, VBN, VBG, VBD	right := $N/PP : \lambda f.\lambda x.\exists y.\textsc{right-rel}(x,y) \wedge f(y)$
$\lambda f.\lambda g.\lambda x.\exists y.r(x,y) \wedge f(y) \wedge g(x)$	$N\backslash N/PP$	VBN, VBG, VBD	located := $N\backslash N/PP : \lambda f.\lambda g.\lambda x.\exists y.\textsc{located-rel}(x,y) \wedge f(y) \wedge g(x)$
	$N\backslash N/N$	V*, IN, TO	near := $N\backslash N/N : \lambda f.\lambda g.\lambda x.\exists y.\textsc{near-rel}(x,y) \wedge f(y) \wedge g(x)$
	$S\backslash N/N, S\backslash N/PP$	V*	borders := $S\backslash N/N : \lambda f.\lambda g.\lambda x.\exists y.\textsc{border-rel}(x,y) \wedge f(y) \wedge g(x)$
$\lambda f.\lambda g.\lambda x.\exists y.r(x,y) \wedge f(x) \wedge g(y)$	$S/N/N, S/N/PP$	V*, except “to be”	borders := $S/N/N : \lambda f.\lambda g.\lambda x.\exists y.\textsc{border-rel}(x,y) \wedge f(x) \wedge g(y)$
$\lambda x.true$	N	PRP, W*, DT, EX	what := $N : \lambda x.true$
$\lambda f.f$	N/N	JJ*, PRP, W*, DT, RB	the := $N/N : \lambda f.f$
	PP/PP	RB, IN, TO	nearly := $PP/PP : \lambda f.f$
	$PP/N, PP\backslash N, PP\backslash PP$	IN, TO	by := $PP/N : \lambda f.f$
	S/N	V*	name := $S/N : \lambda f.f$
$\lambda f.\lambda g.\lambda x.f(x) \wedge g(x)$	$N\backslash N/N$	IN, TO	of := $N\backslash N/N : \lambda f.\lambda g.\lambda x.f(x) \wedge g(x)$
	$N\backslash N/N, PP_r\backslash PP_r/PP_r,$ $PP_r\backslash N/PP_r, PP_r\backslash PP_r/PP_r$	CC	and := $N\backslash N/N : \lambda f.\lambda g.\lambda x.f(x) \wedge g(x)$
	$S\backslash N/N, S\backslash N/PP,$ $S/N/N, S/PP/N$	V*	borders := $S\backslash N/N : \lambda f.\lambda g.\lambda x.f(x) \wedge g(x)$
	$S\backslash N/N, S\backslash N/PP_r,$ $S/N/N, S/PP_r/N$	forms of “to be”	is := $S\backslash N/PP_r : \lambda f.\lambda g.\lambda x.f(x) \wedge g(x)$

Table 6.3: Part-of-speech heuristics for CCG lexicon induction. The logical form templates on the left contain categories c and relations r that are filled with the triggering word’s lemma, as exemplified in the right column. Each logical form template can be created with different syntactic categories depending on the POS tag of the input word. The lexicon contains two types of prepositional phrases: phrases where the preposition indicates a relation (PP_r), and phrases where the preposition has no semantic content (PP). POS tags followed by an “*” match any tag starting with the given sequence (e.g., NN* matches NNS). Consecutive words tagged as NN* are treated as instances of NN*, allowing both words to instantiate a single lexicon entry (e.g., “north east” → NORTHEAST-REL).

ping entries like $\lambda f.\lambda g.\lambda x.\exists y.r(x, y) \wedge g(x) \wedge f(y)$ to $\lambda f.\lambda g.\lambda x.\exists y.g(x) \wedge f(y)$. This model is also trained using the weakly supervised algorithm. The third model, LSP-F, is LSP trained with full supervision, using the manually annotated semantic parses and logical knowledge bases in the data sets. Given these annotations, training LSP amounts to independently training a semantic parser (using sentences with annotated logical forms, $\{(s^i, \ell^i)\}$) and a set of perceptual classifiers (using environments with annotated logical knowledge bases, $\{(d^i, K^i)\}$). This model measures the performance achievable with LSP given significantly more supervision.

All three variants of LSP were trained using the same hyperparameters. For SCENE, training was performed using subgradient descent with 5 example minibatches and 100 passes over the data with $\lambda = 0.03$. For GEOQA, training was performed using subgradient descent with 8 example minibatches and 100 passes over the data with $\lambda = 0.02$. Empirically, model performance was relatively stable while the regularization parameter $\lambda \leq 0.05$. All experiments use leave-one-environment-out cross-validation to estimate model performance. This procedure holds out each environment in turn, trains each model on the remaining environments, then tests on the held-out environment.

6.3.5 Results

The evaluation considers two prediction problems. The first problem is to predict the correct denotation γ^i for a statement s^i in an environment d^i. A correct prediction on this task corresponds to a correctly-answered question. A weakness of this task is that it is possible to guess the right denotation without fully understanding the language. For example, given a query like “mugs on the table,” it might be possible to guess the denotation based solely on “mugs,” ignoring “table” altogether. The grounding prediction task corrects for this problem. Here, each model predicts a grounding, which is the set of all satisfying assignments to the variables in a logical form. For example, for the logical form $\lambda x.\exists y.\text{LEFT-REL}(x, y) \wedge \text{MUG}(y)$, the grounding is the set of (x, y) tuples for which both $\text{LEFT-REL}(x, y)$ and $\text{MUG}(y)$ return true. Note that, if the predicted seman-

Denotation γ	0 rel.	1 rel.	other	total
LSP-CAT	**0.94**	0.45	**0.20**	0.51
LSP-F	0.89	**0.81**	**0.20**	**0.70**
LSP-W	0.89	0.77	0.16	0.67
Grounding g	0 rel.	1 rel.	other	total
LSP-CAT	**0.94**	0.37	0.00	0.42
LSP-F	0.89	**0.80**	0.00	**0.65**
LSP-W	0.89	0.70	0.00	0.59
% of data	23	56	21	100

(a) Results on the SCENE data set.

Denotation γ	0 rel.	1 rel.	other	total
LSP-CAT	0.22	0.19	0.07	0.17
LSP-F	**0.64**	0.53	**0.21**	0.48
LSP-W	**0.64**	**0.58**	**0.21**	**0.51**
Grounding g	0 rel.	1 rel.	other	total
LSP-CAT	0.22	0.19	0.00	0.16
LSP-F	0.64	0.53	**0.17**	0.47
LSP-W	**0.64**	**0.58**	0.15	**0.50**
% of data	8	72	20	100

(b) Results on the GEOQA data set.

Table 6.4: Denotation and grounding prediction accuracy on the SCENE and GEOQA datasets. LSP-CAT is an ablated version of LSP that only learns categories (similar to Matuszek et al. (Matuszek et al., 2012)), LSP-F is LSP trained with annotated semantic parses and logical knowledge bases, and LSP-W is LSP trained on sentences with annotated denotations. Results are separated by the number of relations in each test natural language statement. The best performance in each column is shown in bold.

tic parse is incorrect, the predicted grounding for a statement may contain a different number of variables than the true grounding; such groundings are incorrect. Figure 6.4 shows model predictions for the grounding task.

Performance on both tasks is measured using exact match accuracy. This metric is the fraction of examples for which the predicted set of entities (be it the denotation or grounding) exactly equals the annotated set. This is a challenging metric, as the number of possible sets grows exponentially in the number of entities in the environment. Say an environment has 5 entities and a logical form has two variables; then there are 2^5 possible denotations and 2^{25} possible groundings. To quantify this difficulty, note that selecting a denotation uniformly at random achieves 6% accuracy on SCENE, and 1% accuracy on GEOQA. Selecting a random grounding achieves 1% and 0% accuracy, respectively.

Table 6.4 shows results for both applications using exact match accuracy. To better understand the performance of each model, each test example is placed into one of three categories based on the number of relations in its annotated logical form. The first two categories show performance on examples with 0 and 1 relations. The "other" category shows performance on examples with more than one relation (very infrequent), or that include quantifiers, comparatives, or other linguistic phenomena not captured by LSP.

The results from these experiments suggest three conclusions. First, modelling relations is important for both applications, as (1) the majority of examples contain relational language, and (2) LSP-W and LSP-F dramatically outperform LSP-CAT on these examples. The low performance of LSP-CAT suggests that many denotations cannot be predicted from only the first noun phrase in a statement, demonstrating that both applications require an understanding of relations. Second, weakly supervised training and fully supervised training perform similarly, with accuracy differences in the range of 3%-6%. The next section further examines the differences between these two training regimes. Finally, when LSP predicts the correct denotation, it does so because it understands the entity referents of every part of the statement. This conclusion is supported by the small performance difference between the denotation and grounding prediction tasks on the 1-relation examples. (Groundings cannot be correctly predicted for many of the "other" examples due to the inclusion of unmodelled linguistic phenomena.)

6.3.6 Error Analysis

The error analysis analyzes the mistakes made by each individual model component to better understand the causes of errors. Table 6.5 shows the accuracy of the semantic parser from each trained model. Each held-out sentence s^i is parsed to produce a logical form ℓ, which is marked correct if it exactly matches the annotated logical form ℓ^i. A correct logical form implies a correct grounding for the statement when the parse is evaluated on the annotated knowledge base. Perfect semantic parsing accuracy is not achievable because some natural language queries include unmodelled phenomena, such as comparatives. These results show that both LSP-W and LSP-F have reasonably accurate semantic parsers, given these limitations. Common mistakes include missing lexicon entries (e.g., "borders" is POS-tagged as a noun, so the GEOQA lexicon

	SCENE	GEOQA
LSP-CAT	0.21	0.17
LSP-W	0.72	0.71
LSP-F	**0.73**	**0.75**
Upper Bound	0.79	0.87

Table 6.5: Accuracy of the semantic parser from each trained model. Upper bound is the highest accuracy achievable without modelling comparatives and other linguistic phenomena not captured by LSP.

does not include a verb for it) and prepositional phrase attachments (e.g., 6th example in Figure 6.4).

Table 6.6 shows the precision and recall of the individual perceptual classifiers. These metrics were computed by comparing the annotated denotation of each predicate in the held-out environment with the model's predictions for the same predicate, treating each entity or entity pair as an independent example for classification. Fully supervised training appears to produce better perceptual classifiers than weakly supervised training; however, this result conflicts with the full system evaluation in Table 6.4, where both systems perform similarly well. There are two causes for this phenomenon: uninformative adjectives and unimportant relation instances.

Uninformative adjective predicates are responsible for the low performance of the category classifiers in SCENE. Phrases like "LCD screen" in this domain are annotated with logical forms such as $\lambda x.\text{LCD}(x) \wedge \text{SCREEN}(x)$. Here, LCD is uninformative, since SCREEN already denotes a unique object in the environment. Therefore, it is not important to learn an accurate classifier for LCD. Weakly supervised training learns that the denotation of LCD includes every object in the environment, yet predicts the correct denotation for $\lambda x.\text{LCD}(x) \wedge \text{SCREEN}(x)$ using its SCREEN classifier. This situation happens less frequently in GEOQA because the objects in this domain typically have one word descriptions.

The discrepancy in relation performance occurs because the relation evaluation weights every relation equally, whereas in reality some relations are more frequent. Furthermore, even within a single relation, each entity pair is not equally important – for example, people tend to ask what is

	SCENE			GEOQA		
Categories	P	R	F1	P	R	F1
LSP-CAT	0.40	0.86	0.55	0.78	0.25	0.38
LSP-W	0.40	0.84	0.54	0.85	**0.63**	0.73
LSP-F	**0.98**	**0.96**	**0.97**	**0.89**	**0.63**	**0.74**
Relations	P	R	F1	P	R	F1
LSP-W	0.40	0.42	0.41	0.34	**0.51**	0.41
LSP-F	**0.99**	**0.87**	**0.92**	**0.70**	0.46	**0.55**
Relations (reweighted)	P	R	F1	P	R	F1
LSP-W	**0.98**	**0.98**	**0.98**	0.86	**0.72**	**0.79**
LSP-F	**0.98**	0.95	0.96	**0.89**	0.66	0.76

Table 6.6: Precision, recall and F-score of the trained perceptual classifiers from each model, measured against the annotated knowledge bases. LSP-CAT is excluded from the relation evaluations, since it does not learn relations. See the text for details of the reweighted relations metric.

in a state, but not what is in an ocean. The reweighted relation metric is designed to account for these factors. This metric uses annotated logical forms containing only one relation, of the form $\lambda x.\exists y.c_1(x) \wedge r(x, y) \wedge c_2(y)$, and measures the accuracy of the classifier for r on the set of x, y pairs such that $c_1(x) \wedge c_2(y)$. These entity pairs are the important ones for the relation classifier to classify correctly, as the other pairs are ruled out by the category classifiers. Table 6.6 shows that, using this metric, both training regimes have similar performance. This result suggests that weakly supervised training adapts LSP's relation classifiers to the relation instances that are empirically important for grounding natural language.

6.4 Discussion

This chapter presents Logical Semantics with Perception (LSP), a model for grounded language acquisition that combines a semantic parser with perceptual classifiers to map natural language statements to their referents in a physical environment. It also presents a weakly-supervised training algorithm for LSP that trains both the semantic parser and perceptual classifiers using statements with annotated denotations, without requiring annotated logical forms or the outputs of individual perceptual classifiers. An experimental evaluation demonstrates that LSP outperforms similar prior work that cannot represent relational language, and that both weakly supervised and fully supervised training result in comparable performance.

The positioning of this chapter according to the axes of variation is as follows:

1. **Learning the knowledge base** – The knowledge base is learned using perceptual classifiers that use features of the environment to produce a knowledge base.
2. **Predicate vocabulary** – The semantic parsers use an open predicate vocabulary derived from the training data by the lexicon induction heuristics.
3. **Semantic parser supervision** – Natural language statements with annotated denotations were used to train both the semantic parser and perceptual classifiers.
4. **Syntactic information** – No syntactic information was used aside from part-of-speech tags.

The perception component in this chapter is closely related to the learned probabilistic database in Chapter 5. $f_{per}(K, d; \theta)$ defines an unnormalized probabilistic database, i.e., a weight for every knowledge base K. However, in this chapter, the weight of each predicate instance depends on features of each entity and entity pair derived from the environment d, whereas the previous chapter treated these feature vectors as latent variables to be learned. The fixed feature computation allows the model in this chapter to be applied to previously unseen entities, unlike the learned entity embeddings in the probabilistic database.

The experience of this chapter suggests that it is currently possible to learn a perceptually-grounded semantics for many other applications using LSP and the training algorithms in this chapter, as long as certain conditions are satisfied. First, there must be a feature representation for entities and entity pairs in the environment that can be used in LSP's perception function. This is a minor condition: other fields of computer science have developed effective feature representations for many kinds of environments, e.g., the HOG features in this chapter for image segments. Thus, it is already possible to apply LSP to many other kinds of environments, such as

3-dimensional space. Second, there must be annotated data, in the form of language/denotation pairs, that can be used for training. Third, environments cannot contain large numbers of entities. This condition is necessary in order to train LSP from denotations due to the ILP inference algorithm within the subgradient computation (Section 6.2.2). The number of variables in the ILP grows quadratically in the number of entities in the environment, so this approach does not scale to environments with large numbers of entities. Fourth, the language's meaning must be representable using only category and relation predicates.

Relaxing each of the three final conditions is an open research problem, though there are reasonable approaches to each one. The second condition can be relaxed by developing algorithms that incorporate unlabeled data into training, e.g., in the form of unlabeled sentences from the web. Unlabeled sentences can be used for training by assuming they express true facts. For example, the sentence "a mug is a cup" expresses a constraint on the denotations of the MUG and CUP predicates across all environments. Other forms of training data, such as image/caption pairs, may also be useful. The third condition can be relaxed by developing better approximate inference algorithms or by clevely engineering the form of the training data to avoid intractable inference. (Note that training from denotations requires solving difficult inference problems, but fully-supervised training does not.) The fourth condition can be relaxed by augmenting LSP's knowledge base with additional learned functions that model additional linguistic phenomena. For example, in order to model the generalized quantifier "biggest," the knowledge base must contain a learned function that orders objects by size. It is reasonable to expect some progress on all three of these problems, suggesting that it will be possible to learn a perceptually-grounded semantics for more applications with more complex environments, large predicate vocabularies, and more difficult linguistic phenomena.

Chapter 7

Conclusion

Understanding natural language is a key problem for artificial intelligence research that has numerous applications such as personal voice assistants, information extraction and other core NLP tasks. However, natural language understanding systems today are developed using extensive human annotation, which limits their use. This thesis explores several ways to learn to understand natural language without expensive human annotation. The thesis statement of this work is:

> Language understanding systems can be constructed for both grounded and ungrounded settings using existing resources with little to no additional human annotation.

I evaluate the thesis statement by developing several algorithms for training language understanding systems in different settings with limited human annotation. Chapter 4 studies the problem of training a semantic parser for a large knowledge base, such as NELL or Freebase. Supervised algorithms for training semantic parsers fail in this setting because the number of predicates in the knowledge base necessitates a large training set. This chapter presents two distantly-supervised algorithms for training semantic parsers that eliminated the requirement for these per-sentence semantic annotations, instead using a large corpus of unlabeled text, a large knowledge base and a syntactic resource. These algorithms train semantic parsers using the assumption that each relation instance in the knowledge base is expressed by at least one sentence in the unlabeled text corpus. An empirical evaluation of these algorithms found that a semantic parser trained using distant supervision outperformed a state-of-the-art relation extraction system. It further found that a joint syntactic and semantic parser predicted logical forms more accurately than pipelined syntax-then-semantics approaches.

Chapter 5 considers the problem of building a language understanding system using a learned knowledge base with an open predicate vocabulary. This problem is motivated by two observations: (1) developing a large knowledge base, such as Freebase or NELL, is labor-intensive, and (2) even the largest knowledge bases today lack coverage due to their closed predicate vocabularies. This chapter presents a language understanding system that can compute denotations for compositional natural language queries, such as "Republican front-runner from Texas." The system uses a rule-based semantic parser to map from natural language text to logical forms whose predicates are derived from the words in the text. The denotations of these predicates are then automatically learned using probabilistic matrix factorization with a novel objective function. This objective trains the system to rank correct entity answers to natural language queries above unobserved answers. An experimental evaluation found that this approach outperformed several existing approaches to learning a knowledge base, and furthermore is capable of answering many natural language queries that Freebase cannot.

Finally, Chapter 6 considers the problem of training language understanding systems for grounded settings, where the language understanding agent is situated in a physical environment. This setting is applicable to robots and other autonomous agents. This chapter presents the Logical Semantics with Perception (LSP) model for grounded language understanding, which augments the ungrounded language understanding framework from previous chapters with a perception function that predicts a knowledge base for an environment. This chapter also presents a weakly-supervised algorithm for training LSP using sentences with annotated denotations. These examples represent a natural form of supervision that could be obtained by, for example, follow-

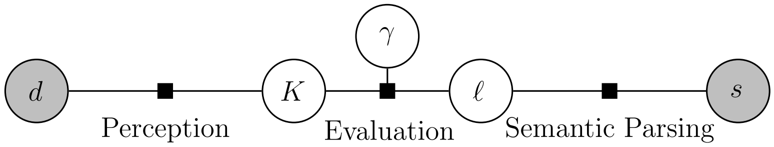


Figure 7.1: Factor graph for language understanding systems. The input to a language understanding system is an environment d and a natural language sentence s, from which the model predicts a knowledge base K, logical form ℓ and denotation γ. Each labeled factor corresponds to a subproblem of language understanding.

ing a user's pointing gesture while they describe an object. An experimental evaluation demonstrated that LSP trained with this weakly-supervised algorithm performs comparably to LSP trained with a fully-supervised algorithm while requiring considerably less human annotation.

There are several other lessons from the algorithms and experiments in this thesis that apply to language understanding systems more generally. The following sections discuss some of these lessons and propose directions for future work.

7.1 Lessons

7.1.1 Framework for Language Understanding

The factor graph in Figure 7.1 is a general framework for thinking about language understanding problems. The goal of language understanding is to predict a denotation γ for a natural language sentence s in an environment d. This prediction problem factors into three subproblems: perception, semantic parsing, and evaluation. This factorization into three subproblems is a natural consequence of the fact that language understanding systems must model both the compositional semantics of language and the mapping between language and an environment. The semantic parser models compositional semantics, perception models the language/environment mapping, and evaluation combines these two components to produce a denotation.

There are two notable aspects of this language understanding framework. First, the central problem is to predict the denotation of a sentence. This focus is a natural consequence of this framework's origins in model-theoretic or truth-conditional semantics (Dowty et al., 1981). This choice is motivated by two considerations: (1) understanding problems often require the ability to map from natural language to non-linguistic entities, and (2) many other tasks, such as textual entailment and question answering, can naturally be formulated as reasoning about denotations. Mapping language to non-linguistic entities is required for both perceptually-grounded understanding (e.g., a robot) and virtual agents (e.g., a mobile phone voice assistant). Further note that, by conditioning on the denotation, this framework naturally enables inferences about the real world (represented by the environment d) given a sentence. Therefore, another perspective on this framework is that language understanding is fundamentally about *mapping between language and the real world*.

Second, the semantic representation for logical forms ℓ and the range of values for denotations γ are intentionally underspecified. The only requirement for the semantic representation is that it can be (perhaps nondeterministically) evaluated using knowledge about the environment K. Denotations throughout this thesis have been either sets of entities or truth values, but other values are also possible, such as paths through an environment (Kollar et al., 2010). In my view, the semantic representation is a *means* to the end of predicting denotations: what ultimately matters is whether the question is answered correctly, or the robot performs the right action. Thus, the semantic representation is best viewed as a modelling choice that impacts the language understanding model's expressivity and learnability. The logical language used throughout this thesis is a reasonable semantic representation, but it has several limitations; for example, the set of entities must be fixed beforehand and learning sometimes requires solving hard inference problems. I anticipate that designing new semantic representations – using various denotation prediction tasks for evaluation – will be a productive area for future work (see Section 7.2.1).

It is instructive to compare this framework for language understanding with other perspectives on natural language semantics. The remainder of this section compares and contrasts these perspectives.

Frame Semantics

Frame semantics is a theory of natural language semantics where the meaning of a natural language sentence is represented by a collection of *frames*. A frame is a description of a stereotypical relation, entity or event and its participants. For example, the Commercial Transaction Frame describes buying and selling goods, and its participants include a buyer, a seller, an item and a monetary amount. Although the theory does not inherently specify a vocabulary of frames, the FrameNet (Baker et al., 1998) frame vocabulary is typically used in practice.

Frame semantics is best viewed as a choice of a semantic representation language, i.e., a formalism for defining logical forms ℓ. This representation is in many ways similar to the logical language used throughout this work because frames can be viewed as predicates. However, frame semantics underspecifies certain aspects of meaning (notably quantifiers); consequently frame semantic representations do not have a natural evaluation function. Rather, frame semantics is better viewed as a way of collapsing roughly synonymous natural language constructions to the same semantic representation. Of course, it may still be possible to construct an evaluation function for a frame semantic representation using machine learning.

Abstract Meaning Representation (AMR)

Abstract Meaning Representation (AMR) is a graph-based semantic representation designed to provide a common formalism for all kinds of semantic annotations Banarescu et al. (2013). Thus, AMR annotations incorporate coreference, semantic relations (based on PropBank Kingsbury and Palmer (2002); Palmer et al. (2005)), discourse connectives, and temporal information. An AMR annotation is most commonly viewed as a directed, rooted graph describing the entities, relations and events in the sentence. However, every AMR graph to be converted into a equivalent logical statement similar to those used throughout this thesis.

As with frame semantics, AMR is a semantic representation language. However, unlike frame semantic representations, AMR annotations can be evaluated due to the equivalence between AMR graphs and logical statements. (Note that evaluating AMR requires a knowledge base, which can perhaps be learned as in Chapter 5). Furthermore, AMR is more expressive than the existentially-quantified conjunctions used throughout this thesis: it can represent modals, negation, coreference, and cross-sentence phenomena. AMR may be a good representation language for ungrounded tasks that do not require the use of a specific predicate vocabulary (as in Chapter 5). It may also be possible to design variants of AMR with different sets of predicates for use in other language understanding applications.

Vector Space Semantics / Neural Networks

Vector space semantics treats the meaning of a natural language word or phrase as a vector in some high-dimensional space. This perspective does not necessarily impose a fixed interpretation of this vector nor a method for producing it. Consequently, this approach to semantics is quite flexible and the properties of the vector representation depend on the particulars of the application.

It is useful to categorize work in vector space semantics into two perspectives. The *distributional* perspective argues that the meaning of a word is best described by the distribution of its coocurrences with other words or documents Firth (1957). These distributional representations are empirically quite useful for representing the semantics of individual words Turney and Pantel (2010), leading to the question of how to compose them to produce phrase and sentence representations. Work using this perspective tends to learn word representations by counting cooccurrences in an unsupervised way, typically with fixed composition functions (Coecke et al., 2010; Grefenstette and Sadrzadeh, 2011). This perspective is best viewed as an alternative theory of semantics where coocurrence is the primary modelling consideration. While this perspective has been useful for some applications – specifically, tasks that require computing word similarity – it is unclear how to use these models to predict denotations (see the discussion below). However, the distributional information captured by this perspective is quite valuable and can be incorporated into other kinds of language understanding models (as in Chapter 5).

The *prediction* perspective argues that vectors are a flexible representation for encoding arbitrary semantic information, so predictive models that use vectors as their semantic representation can be trained to perform well on many applications. Neural network models tend to rely on this perspective (e.g., Socher et al. (2011)). Note that distributional information is often included within these predictive models (e.g., using unsupervised pretraining). This perspective's focus on prediction aligns with the framework of this thesis, where the goal is to predict denotations. Although no vector space model has yet been constructed for a denotation prediction task of similar difficulty as those in this thesis, such a model seems possible and may have some advantages over the approaches in this thesis due to the flexibility of vector representations.

Which components of the language understanding framework can be represented as vectors? Generally, it seems unlikely that a vector can take the place of a logical form. For a vector representation to serve this purpose, it would have to be evaluatable, thereby representing a function from all possible environments to denotations. This mapping can require reasoning about complex structure in the environment, for example, if the logical form contains a number of relations and quantifiers. However, it does seem possible for vectors to encode denotations themselves (e.g., sets of entities) because denotations are composed using simple operations (e.g., intersection) according to local sentence structure. The environment could be incorporated into such a vector space model at the level of individual words, allowing each word's vector in the sentence to represent its denotation. Semantic parsing may still play a role within these models to determine the structure of the vector space composition. Such vector space models are a promising area for future work (see Section 7.2.1).

Textual Entailment

Textual entailment is the problem of computing entailment relationships between a text and a hypothesis Dagan et al. (2006). A text entails a hypothesis if, under all possible worlds where the text is true, the hypothesis is also true. For example, an entailment system should predict that the text “Google files for its IPO” entails the hypothesis “Google goes public.” Note that other tasks in semantics, such as question answering, can be reformulated as textual entailment questions.

Textual entailment is best understood as an *evaluation task* for language understanding systems. A key motivation for this task is its representation independence: any semantic representation can be used as an intermediate representation in the process of making entailment judgments. Thus, this task enables the comparison of different semantic representations on a common inference problem. I argued above that such comparison is an essential next step in developing effective semantic representations for language understanding, and textual entailment is a natural task for performing this comparison (see Section 7.2.3).

Textual entailment systems cannot necessarily be used for language understanding because these systems are not necessarily able to reason about non-linguistic entities. This limitation prevents such systems from being applied to tasks that inherently require such reasoning; for example, commanding a robot requires mapping from natural language to objects and locations in the environment. Note, however, that it is possible to construct a textual entailment system by augmenting the language understanding framework of this thesis with a distribution over environments. Entailment judgments can then be made by conditioning the environment distribution on the text being true, then computing the probability that the hypothesis is true under this distribution. Thus, language understanding can be viewed as a subtask of textual entailment, but not vice versa.

7.1.2 Unlabeled Text as Training Data

Unlabeled text is a key source of training data in this thesis. The common assumption made when using unlabeled text is that sentences express true statements, i.e., the denotation of a sentence is TRUE. This assumption provides a common starting point for using unlabeled text to train any language understanding system. Developing a training algorithm amounts to adapting this assumption to suit the particular system being trained. For example, the query ranking objective in Chapter 5 trains the probabilistic database to assign a higher truth probability to an observed sentence than a random, unobserved sentence. The distant supervision procedure in Chapter 4 uses this assumption in the distant supervision constraint that forces the predicted logical forms of unlabeled sentences to entail relation instances in the knowledge base. This assumption can presumably be adapted in other ways to train more complex language understanding systems.

Syntactic information seems to be required in order to effectively use unlabeled text to train a language understanding system. Syntactic information is necessary because most naturally-occurring sentences (e.g., on the web) have complex syntactic structures that are difficult to semantically parse correctly without some prior knowledge of syntax. This result is unfortunate, as syntactic treebanks are themselves quite expensive to annotate. However, syntactic treebanks already exist for many languages and they are a valuable investment given the wide applicability of syntactic parsing to downstream tasks. In contrast, a corpus of semantically-annotated sentences is less reusable as differences between semantic representations can make the annotations inapplicable. Therefore, training language understanding systems using syntactic resources still represents an overall reduction in human annotation effort.

7.1.3 Learning a Knowledge Base

A key idea in this thesis is to *learn* the knowledge base used by the language understanding system. This learning problem is encapsulated in the perception factor in Figure 7.1. This idea represents a significant departure from prior work on semantic parsing which assumes that a fixed knowledge base is given to the system. Learning the knowledge base enables the system to use an open predicate vocabulary, which in turn improves the system's coverage. It also enables the knowledge base to depend on an environment, which is necessary for grounded applications.

Learning a knowledge base can be naturally posed as binary classification, where per-predicate classifiers are trained to predict which entities are elements of each predicate's denotation. This framing of the learning problem works for both category and relation predicates. An observation from this thesis is that training these per-predicate classifiers is easier in the grounded setting (Chapter 6), as feature vectors for entities can be deterministically derived from the environment. In the ungrounded setting (Chapter 5), the features of each entity are parameters that must be estimated during training. An open question is how to learn more complex kinds of knowledge base elements, such as n-ary predicates for $n > 2$, or generalized quantifiers (e.g., BIGGEST).

7.1.4 Intractable Inference

Intractable inference problems frequently arise when training language understanding systems. There are two common intractable inference problems. The first is finding the best parse of a sentence that produces a particular logical form or denotation. This inference problem is prevalent in most work on semantic parsing, where it is typically approximately solved using beam search (see Section 2.2.5). However, this inference problem becomes even more challenging when the sentences increase in length (Chapter 4) or when the knowledge base is a random variable (Chapter 6). Beam search still works in these regimes, but may require additional heuristics to perform well.

The second intractable inference problem is, given a logical form ℓ and denotation γ, to find the highest-scoring knowledge base where ℓ evaluates to γ. This problem can arise when training a perception function. Whether or not this problem is intractable depends on the structure of the logical form ℓ. In this thesis, the main cause of intractable inference is existentially-quantified variables in ℓ that couple the boolean membership variables for each entity in the denotation. For example, consider the distribution over denotations for the logical form $\lambda x.\exists y.r(x, y) \wedge c(y)$. The distribution over knowledge bases contains a boolean random variable $c_2(e)$ for each entity e, and the probability that every entity x is an element of the denotation depends on *all* of these variables. This problem occurs in the training procedure for LSP, necessitating the integer linear program inference algorithm. However, it is also possible to formulate the training problem to avoid intractable inference. The training data generation procedure for the query ranking objective (Chapter 5) is specifically designed to generate logical forms where inference is tractable.

7.2 Future Work

7.2.1 Improved Machine Learning Models for Language Understanding

This thesis approached the problem of language understanding by building machine learning models that implement a simple model-theoretic semantics. These models can be improved in several ways.

The first direction for further research is to improve the expressivity of these models to represent more complex linguistic phenomena. In the ungrounded setting, a useful extension is to represent events and time. These phenomena are important for information extraction, and seem particularly well-suited to being extracted by semantic parsing. Another similar extension is to model generalized quantifiers, such as counting and superlatives. Generalized quantifiers are also interesting in the grounded setting, though this problem is more challenging because each generalized quantifier has a domain-specific interpretation that must be learned.

A second direction is to explore new semantic representations that are explicitly designed for machine learning. The view of this thesis is that the fundamental problem in language understanding is predicting the denotation of a given text (as in Figure 7.1). A semantic representation defines the structure of a machine learning model for this denotation prediction problem by specifying the space of possible logical forms and knowledge bases. Crucially, the choice of semantic representation also determines the difficulty of each of the three subproblems. This thesis provides several examples of different representations resulting in very different learning problems. For example, in the setting with a given knowledge base (Chapter 4), perception is trivial but semantic parsing is challenging. In the setting with a learned knowledge base with an open predicate vocabulary (Chapter 5), the difficulty of these problems is reversed. Therefore, it is natural to ask whether some representations lead to better predictions or perhaps permit tractable inference.

Within this second direction, vector space semantics or neural networks provide a particularly interesting class of semantic representations. These approaches represent the meaning of language as a vector in a high-dimensional space and perform composition using (possibly learned) mathematical operations on these vectors. This approach contrasts with the logical view used in this thesis, where the meaning of language is represented as a lambda calculus statement and composition is performed by semantic parsing. However, these two approaches can be combined: Chapters 5 and 6 both trained collections of entity classifiers for predicates in the knowledge base, the parameters of which can be viewed as vector representations of word meanings. The question is thus whether vector space composition can be used to accurately predict denotations. Vector space composition seems better suited for some phenomena, such as slightly noncompositional word meanings, but worse for others. In either case, these vector space models have the significant advantage of always permitting tractable inference.

A final direction is to incorporate machine learning directly into theories of model-theoretic semantics to extend their expressivity. Such theories could incorporate probabilistic constructions that could be useful for modeling possible worlds, hypotheticals and counterfactuals, all of which seem to inherently depend on the language understanding agent's predictions about its environment. Another area where machine learning can help is in relaxing the assumptions made by such theories, especially about the language understanding agent's representation of the world. Here, it is typical to assume that the agent has access to a knowledge base containing all of the world's entities and knowledge. However, many of these assumptions are unreasonable. For example, it is unreasonable to assume that all of the world's entities are given to the agent, as it is easy to name new entities that no person has previously thought of. It may be possible to formulate new theories that make weaker assumptions by assuming the agent has access to certain (machine-learnable) functions that can, for example, identify or invent new entities.

7.2.2 Improved Algorithms for Semantic Parsing

A second direction for future work is improved algorithms for two subproblems of semantic parsing. The first problem is finding the highest-scoring CCG parse conditioned on a logical form. This inference problem is critical for training a semantic parser, yet no exact algorithm is known; instead, the current best practice is to perform beam search. However, beam search can be slow even with small grammars, and often fails to find a solution with more complex grammars. There are two possible approaches for developing a better inference algorithm. The first approach is to extend algorithms from syntactic CCG parsing for finding the highest-scoring CCG parse that produces a particular collection of dependency structures. The second approach is to use properties of the particular logical forms in the lexicon to intelligently guide the beam search – for example, to detect early in the search that a particular logical form can be discarded because it contains a predicate that does not appear in the conditioned on logical form.

The second problem is inducing a lexicon for a semantic parser. Although several approaches to lexicon induction exist, they have important limitations. The first approach, used in this thesis, is to induce the lexicon using a collection of manually-engineered heuristics. However, this requires human effort and may result in low coverage. A second approach is to perform a search over lexicons while training the semantic parser. However, these algorithms require annotated logical forms as input and cannot be used with weaker labels such as denotations or distant supervision. Furthermore, it is difficult to analyze the properties of these algorithms. Developing more principled and flexible approaches to lexicon induction is an area for future work.

7.2.3 Grounded Knowledge Representations

A final direction for future work is to develop broad-coverage *grounded* knowledge representations for language understanding. A grounded knowledge representation is, essentially, a trained perception model in the language understanding framework depicted in Figure 7.1. As an example, LSP (Chapter 6) learned a grounded knowledge representation consisting of a collection of category and relation classifiers for image segments. These representations can be substituted for logical knowledge bases in many tasks with several advantages. First, the perceptual classifiers in grounded representations implicitly encode complex properties of predicates. For example, the LSP classifier for MUG encodes shape, size, and color information. Many properties that can be encoded by such classifiers are difficult to represent in logical formalisms. Second, grounded representations enable inference via reasoning over possible worlds. For example, LSP could answer the question "are all mugs cups?" by searching a large image collection for all image segments depicting mugs, then determining if those image segments also depict cups. Intuitively, this question asks, in all possible worlds, can every object called "mug" also be called "cup?" The proposed inference algorithm directly tackles this question using a corpus of images to approximate the distribution over possible worlds and the grounded knowledge representation to identify mugs and cups. Reasoning over possible worlds in this fashion naturally incorporates the complex correlations of the real world; such correlations are difficult, if not impossible, to represent using logical rules.

There are several research directions for grounded knowledge representations. The first is to develop algorithms for mapping from language to grounded representations of various concrete domains, such as 3-dimensional space and images. Initial work in this area should build on existing grounded representations developed in other fields of computer science, such as robotics and vision. Interestingly, these fields have grounded representations for some domains that are classically difficult to represent in logic. For example, inverse reinforcement learning could be used to model the goals, beliefs and desires of agents (Ng and Russell, 2000). More abstract domains, such as social relationships, may be representable using metaphors from concrete domains (Lakoff and Johnson, 1980). It is important to develop algorithms for multiple domains, as each domain will present unique modeling challenges. Identifying commonalities between the various models will then enable us to work toward a unified framework for learning to understanding language in grounded settings.

A second direction is to build large grounded knowledge representations that have sufficient coverage to be used in downstream inference tasks. However, annotating training data for these grounded knowledge representations will be a major challenge. Thus, a particularly interesting direction is to populate these representations by reading unstructured text. I call this problem *grounded information extraction*. Algorithms for grounded information extraction will combine the grounded language understanding models developed above with traditional information extraction techinques, such as bootstrapping (Riloff and Jones, 1999), to populate large grounded knowledge representations. A promising first step in this direction is to extend ImageNet (Deng et al., 2009) with new categories by automatically reading object descriptions.

A final direction is to apply grounded knowledge bases to difficult natural language inference tasks, such as textual entailment. The first step in this direction is to consider simplifications of the inference task that rely on a particular kind of knowledge, such as visual knowledge. These tasks could be used to develop inference algorithms (using the possible worlds technique described above) and to compare against text-only approaches. The success of these inference algorithms will largely depend on having accurate, high-coverage grounded knowledge representations; ideally, these representations will be produced by grounded information extraction. As the sophistication of modeling techniques improves, it will be possible to combine various kinds of knowledge and perform more general inference tasks.

Bibliography

Kazimierz Ajdukiewicz. Die syntaktische konnexitt. *Studia Philosophica*, 1:1–27, 1935. 16

Jacob Andreas and Dan Klein. Grounding language with points and paths in continuous spaces. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, 2014. 52

Jacob Andreas, Andreas Vlachos, and Stephen Clark. Semantic parsing as machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2013. 34

Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1(1): 49–62, 2013. 9, 10, 34

Yoav Artzi, Dipanjan Das, and Slav Petrov. Learning compact lexicons for CCG semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. 10, 34, 43

Michael Auli and Adam Lopez. Training a log-linear parser with loss functions via softmax-margin. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, 2011a. 10, 45

Michael Auli and Adam Lopez. Efficient CCG parsing: A* versus adaptive supertagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, 2011b. 45

Michael Auli and Adam Lopez. A comparison of loopy belief propagation and dual decomposition for integrated CCG supertagging and parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011c. 45

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. The Berkeley FrameNet project. In *Proceedings of the 17th International Conference on Computational Linguistics - Volume 1*, 1998. 153

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, 2013. 153

Srinivas Bangalore and Aravind K. Joshi. Supertagging: an approach to almost parsing. *Computational Linguistics*, 25(2):237–265, 1999. 21, 45

Michele Banko and Oren Etzioni. The tradeoffs between open and traditional relation extraction. In *Proceedings of ACL-08: HLT*, 2008. 49

Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, 2007. 5, 49, 98

Yehoshua Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, pages 47–58, 1953. 16

Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014. 3, 7, 34, 35, 36, 37, 98, 119

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013. 7, 33, 36, 37, 98, 114

Chris Biemann. Chinese whispers: An efficient graph clustering algorithm and its application to natural language processing problems. In *Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing*, 2006. 115

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008. 3, 39, 56

Benjamin Börschinger, Bevan K. Jones, and Mark Johnson. Reducing grounded learning tasks to grammatical inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011. 39

Johan Bos. Towards wide-coverage semantic interpretation. In *In Proceedings of Sixth International Workshop on Computational Semantics IWCS-6*, 2005. 46, 88

Johan Bos. Wide-coverage semantic analysis with boxer. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, 2008. 46, 104

Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING '04)*, 2004. 46

Razvan C. Bunescu and Raymond J. Mooney. A shortest path dependency kernel for relation extraction. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 2005. 47

Razvan C. Bunescu and Raymond J. Mooney. Learning to extract relations from the web using minimal supervision. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, 2007. 40, 48

Qingqing Cai and Alexander Yates. Semantic parsing Freebase: Towards open-domain semantic parsing. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics (*SEM)*, 2013a. 10, 34, 40

Qingqing Cai and Alexander Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the Annual Meeting of the Association for Computational*

Linguistics (ACL), 2013b. 3, 7, 10, 34, 40, 98, 114

Rachael Cantrell. Mink: An incremental data-driven dependency parser with integrated conversion to semantics. In *Proceedings of the Student Research Workshop*, 2009. 33

Rehj Cantrell, Matthias Scheutz, Paul Schermerhorn, and Xuan Wu. Robust spoken instruction understanding for HRI. In *Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction*, 2010. 52

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010. 5, 39, 56

David L. Chen and Raymond J. Mooney. Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th International Conference on Machine learning*, 2008. 37, 39

David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, 2011. 9, 36, 37

Janara Christensen, Mausam, Stephen Soderland, and Oren Etzioni. Semantic role labeling for open information extraction. In *Proceedings of the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading*, 2010. 49

Stephen Clark. Supertagging for Combinatory Categorial Grammar. In *Proceedings of the International Workshop on Tree Adjoining Grammars and Related Frameworks*, 2002. 21

Stephen Clark and James R. Curran. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552, 2007a. 10, 11, 20, 28, 44, 77, 78, 81, 82, 87, 88

Stephen Clark and James R. Curran. Perceptron training for a wide-coverage lexicalized-grammar parser. In *Proceedings of the Workshop on Deep Linguistic Processing*, 2007b. 44

Stephen Clark and James R. Curran. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of the 20th International Conference on Computational Linguistics*, 2004. 21, 45

Stephen Clark, Julia Hockenmaier, and Mark Steedman. Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002. 44

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving semantic parsing from the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, 2010. 34, 36

Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical Foundations for a Compositional Distributed Model of Meaning. *Lambek Festschirft, Linguistic Analysis*, 36, 2010. 154

Michael Collins. Three generative, lexicalized models for statistical parsing. In *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, 1997.

32, 44

Michael Collins. Discriminative training methods for Hidden Markov Models: theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2002. 26, 27, 65

Mark Craven and Johan Kumlien. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, 1999. 40, 48

Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, 2004. 47

Ido Dagan, Oren Glickman, and Bernardo Magnini. The PASCAL recognising textual entailment challenge. In *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, 2006. 155

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005. 138

Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4), 2007. 108

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977. 37

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, 2009. 163

George Doddington, Alexis Mitchell, Mark Przybocki, Lance Ramshaw, Stephanie Strassel, and Ralph Weischedel. The automatic content extraction (ACE) program tasks, data, and evaluation. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, 2004. 47

David R. Dowty, Robert E. Wall, and Stanley Peters. *Introduction to Montague Semantics*. 1981. 2, 14, 152

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. 116

Juraj Dzifcak, Matthias Scheutz, Chitta Baral, and Paul Schermerhorn. What to do and how to do it: translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, 2009. 52

Jason Eisner. Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 1996. 20

Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen

Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, June 2005. 49

Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011. 49

John R. Firth. *Papers in Linguistics, 1934-1951.* Oxford University Press, 1957. 154

Timothy A. D. Fowler and Gerald Penn. Accurate context-free parsing with Combinatory Categorial Grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 2010. 19, 45

Evgeniy Gabrilovich, Michael Ringgaard, and Amarnag Subramanya. FACC1: Freebase annotation of ClueWeb corpora, Version 1 (Release date 2013-06-26, Format version 1, Correction level 0), 2013. http://lemurproject.org/clueweb09/. 103, 113

Ruifang Ge and Raymond J. Mooney. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, 2005. 32

Ruifang Ge and Raymond J Mooney. Learning a compositional semantic parser using an existing syntactic parser. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, 2009. 9, 33, 43

Dan Goldwasser, Roi Reichart, James Clarke, and Dan Roth. Confidence driven unsupervised semantic parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, 2011. 34, 41

Edward Grefenstette and Mehrnoosh Sadrzadeh. Experimental support for a categorical compositional distributional model of meaning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011. 154

Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 2*, 1992. 49

Julia Hockenmaier. Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, 2003a. 10, 44

Julia Hockenmaier. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh, 2003b. 87, 88

Julia Hockenmaier and Mark Steedman. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of Third International Conference on Language Resources and Evaluation*, 2002a. 11, 28, 43, 57, 77

Julia Hockenmaier and Mark Steedman. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002b. 10, 44

Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke S. Zettlemoyer, and Daniel S. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In

The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, 2011. 40, 41, 48, 62, 64, 66, 68, 86

Thomas Howard, Stefanie Tellex, and Nicholas Roy. A natural language planner interface for mobile manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2014)*, 2014. 52

Hans Kamp and Uwe Reyle. *From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer, 1993. 46

Rohit J. Kate and Raymond J. Mooney. Learning language semantics from ambiguous supervision. In *Proceedings of the 22nd Conference on Artificial Intelligence*, 2007. 37, 39

Rohit J. Kate and Raymond J. Mooney. Using string-kernels for learning semantic parsers. In *21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, 2006. 33, 35

Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. Learning to transform natural to formal languages. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, 2005. 32, 35

Joohyun Kim and Raymond J. Mooney. Generative alignment and semantic parsing for learning from ambiguous supervision. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, 2010. 39

Joohyun Kim and Raymond J. Mooney. Unsupervised PCFG induction for grounded language learning with highly ambiguous supervision. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and Natural Language Learning (EMNLP-CoNLL '12)*, 2012. 37, 39

Joohyun Kim and Raymond J. Mooney. Adapting discriminative reranking to grounded language learning. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013. 37

Paul Kingsbury and Martha Palmer. From treebank to propbank. In *Proceedings of the Third International Conference on Language Resources and Evaluation, LREC 2002, May 29-31, 2002, Las Palmas, Canary Islands, Spain*, 2002. 153

Dan Klein and Christopher D. Manning. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, 2003. 45

Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction*, 2010. 52, 152

Thomas Kollar, Jayant Krishnamurthy, and Grant Strimel. Toward interactive grounded language acquisition. In *Proceedings of Robotics: Science and Systems*, 2013. 127

Jayant Krishnamurthy and Thomas Kollar. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association of Computational Linguistics – Volume 1*, pages 193–206, 2013. 10, 34, 127

Jayant Krishnamurthy and Tom M. Mitchell. Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012. 10, 23, 34, 41, 57, 98

Jayant Krishnamurthy and Tom M. Mitchell. Vector space semantic parsing: A framework for compositional vector space models. In *Proceedings of the 2013 Workshop on Continuous Vector Space Models and their Compositionality*, 2013. 121

Jayant Krishnamurthy and Tom M. Mitchell. Joint syntactic and semantic parsing with Combinatory Categorial Grammar. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014. 10, 34, 41, 57

Jayant Krishnamurthy and Tom M. Mitchell. Learning a compositional semantics for Freebase with an open predicate vocabulary. *Transactions of the Association of Computational Linguistics*, 2015. 100

Marco Kuhlmann and Giorgio Satta. A new parsing algorithm for Combinatory Categorial Grammar. *Transactions of the Association for Computational Linguistics*, 2(Oct):405–418, 2014. 19

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010. 10, 34, 35, 42, 43

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011. 7, 10, 34, 40, 51

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013. 7, 10, 34, 35, 98

George Lakoff and Mark Johnson. *Metaphors we Live by*. University of Chicago Press, 1980. 163

Kenton Lee, Yoav Artzi, Jesse Dodge, and Luke Zettlemoyer. Context-dependent semantic parsing for time expressions. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014. 10, 34

Douglas B Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995. 3

Mike Lewis and Mark Steedman. Combined distributional and logical semantics. *Transactions of the Association for Computational Linguistics*, 1:179–192, 2013. 46, 104, 115

Mike Lewis and Mark Steedman. A* CCG parsing with a supertag-factored model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. 10, 45

Percy Liang, Michael I. Jordan, and Dan Klein. Learning dependency-based compositional semantics. In *Proceedings of the Association for Computational Linguistics*, 2011. 7, 33, 36, 37, 38

Percy Liang, Michael I. Jordan, and Dan Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446, June 2013. 33

Dekang Lin and Patrick Pantel. DIRT — discovery of inference rules from text. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001. 50, 115

Wei Lu, Hwee Tou Ng, Wee Sun Lee, and Luke S. Zettlemoyer. A generative model for parsing natural language to meaning representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2008. 33, 35, 39

Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: connecting language, knowledge, and action in route instructions. In *AAAI*, 2006. 33

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999. 19

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. 114

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993. 44

Cynthia Matuszek, Dieter Fox, and Karl Koscher. Following directions using statistical machine translation. In *Proceedings of the 5th ACM/IEEE International Conference on Human-Robot Interaction*, 2010. 9, 36

Cynthia Matuszek, Nicholas FitzGerald, Luke Zettlemoyer, Liefeng Bo, and Dieter Fox. A joint model of language and perception for grounded attribute learning. In *Proceedings of the 29th International Conference on Machine Learning*, 2012. 10, 34, 51, 127, 136, 140, 143

Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012. 50

George A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38 (11):39–41, November 1995. 42

David Milne and Ian H. Witten. Learning to link with Wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, 2008. 103

Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013. 48

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 2009. 40, 48, 68

Andrew Y Ng and Stuart J Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, 2000. 162

Truc-Vien T. Nguyen and Alessandro Moschitti. End-to-end relation extraction using distant supervision from external semantic repositories. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, 2011. 48

Joakim Nivre, Johan Hall, and Jens Nilsson. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, 2006. 45, 67

Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, 2003. 45

Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003. 43

Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, March 2005. 153

Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. Web-scale distributional similarity and entity set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, 2009. 42

Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, 2007. 45

Hoifung Poon. Grounded unsupervised semantic parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2013. 41

Hoifung Poon and Pedro Domingos. Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 2009. 115

Lev Ratinov, Dan Roth, Doug Downey, and Mike Anderson. Local and global algorithms for disambiguation to Wikipedia. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011. 103

Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. (online) subgradient methods for structured prediction. *Artificial Intelligence and Statistics*, 2006. 85, 133

Siva Reddy, Mirella Lapata, and Mark Steedman. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association of Computational Linguistics – Volume 2, Issue 1*, 2014a. 98

Siva Reddy, Mirella Lapata, and Mark Steedman. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392, 2014b. 11, 34, 40

Terry Regier and Laura A. Carlson. Grounding spatial language in perception: An empricial and computational investigation. *Journal of Experimental Psychology: General*, 130(2):273–298, 2001. 138

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR:

Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 2009. 111

Sebastian Riedel, Limin Yao, and Andrew McCallum. Modeling relations and their mentions without labeled text. In *Proceedings of the 2010 European conference on Machine learning and Knowledge Discovery in Databases*, 2010. 40, 48, 62

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. Relation extraction with matrix factorization and universal schemas. In *Joint Human Language Technology Conference/Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, 2013. 5, 50, 98, 111, 121

Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, 1999. 163

Alan Ritter, Luke Zettlemoyer, Mausam, and Oren Etzioni. Modeling missing data in distant supervision for information extraction. *Transactions of the Association of Computational Linguistics – Volume 1*, pages 367–378, 2013. 49

Dan Roth and Wen-tau Yih. Global inference for entity and relation identification via a linear programming formulation. pages 553–580, 2007. 34

Deb K Roy. Learning visually grounded words and syntax for a scene description task. *Computer Speech & Language*, 16(3):353–385, 2002. 51

Nobuyuki Shimizu and Andrew Haas. Learning to follow navigational route instructions. In *Proceedings of the 21st international joint conference on artifical intelligence*, 2009. 36

Marjorie Skubic, Dennis Perzanowski, Sam Blisard, Alan Schultz, William Adams, Magda Bugajska, and Derek Brock. Spatial language for human-robot dialogs. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(2):154–167, 2004. 52

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011. 121, 155

Robert Speer, Catherine Havasi, and Henry Lieberman. AnalogySpace: Reducing the dimensionality of common sense knowledge. In *AAAI*, 2008. 50

Mark Steedman. *The syntactic process*. MIT Press, Cambridge, MA, USA, 2000. 10

Mark Steedman. *Surface Structure and Interpretation*. The MIT Press, Cambridge, MA, USA, 1996. 76

Mark Steedman and Jason Baldridge. Combinatory categorial grammar. *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, pages 181–224, 2011. 10, 11

Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases. *Synthesis Lectures on Data Management*, 3(2):1–180, 2011. 101, 109

Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D. Manning. Multiinstance multi-label learning for relation extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012. 48

Shingo Takamatsu, Issei Sato, and Hiroshi Nakagawa. Reducing wrong labels in distant supervision for relation extraction. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, 2012. 48

Lappoon R. Tang and Raymond J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*, 2001. 32

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Advances in Neural Information Processing Systems*. 2004. 133

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI Conference on Artificial Intelligence*, 2011. 52

Stefanie Tellex, Pratiksha Thaker, Joshua Joseph, and Nicholas Roy. Learning perceptually grounded word meanings from unaligned parallel data. *Machine Learning*, 94(2):151–167, 2014. 52

Ivan Titov and Alexandre Klementiev. A Bayesian model for unsupervised semantic parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, 2011. 115

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2003. 140

Peter D. Turney. The latent relation mapping engine: Algorithm and experiments. *Journal of Artificial Intelligence Research*, 33(1):615–655, December 2008. 50

Peter D. Turney and Patrick Pantel. From frequency to meaning: vector space models of semantics. *Journal of Artificial Intelligence Research*, 37(1), January 2010. 154

K. Vijay-Shanker and David J. Weir. Polynomial time parsing of Combinatory Categorial Grammars. In *Proceedings of the 28th Annual Meeting on Association for Computational Linguistics*, 1990. 19

K. Vijay-Shanker and David J. Weir. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636, 1993. 19

David J. Weir and Aravind K. Joshi. Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, 1988. 18

Terry Winograd. *Procedures as a representation for data in a computer program for understanding natural language.* PhD thesis, Massachusetts Institute of Technology, 1970. 1

Yuk Wah Wong and Raymond J. Mooney. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL*,

2006. 33, 35

Yuk Wah Wong and Raymond J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, 2007. 9, 33, 35

Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403, 1997. 33, 35

Fei Wu and Daniel S. Weld. Open information extraction using Wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 2010. 40, 49

Fei Wu and Daniel S. Weld. Autonomously semantifying wikipedia. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, 2007. 48

Fei Wu, Raphael Hoffmann, and Daniel S. Weld. Information extraction from Wikipedia: Moving down the long tail. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008. 48

Wei Xu, Raphael Hoffmann, Le Zhao, and Ralph Grishman. Filling knowledge base gaps for distant supervision of relation extraction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2013. 48

Wenduan Xu, Stephen Clark, and Yue Zhang. Shift-reduce CCG parsing with a dependency model. 2014. 45

Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012. 98

Limin Yao, Sebastian Riedel, and Andrew McCallum. Universal schema for entity type prediction. In *Proceedings of the 2013 Workshop on Automated Knowledge Base Construction*, 2013. 50

Alexander Yates and Oren Etzioni. Unsupervised resolution of objects and relations on the web. In *Proceedings of the 2007 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2007. 50

Haonan Yu and Jeffrey Mark Siskind. Grounded language learning from video described with sentences. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2013. 51

Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. Kernel methods for relation extraction. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, 2002. 47

John M. Zelle and Raymond J. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the 11th National Conference on Artificial Intelligence*, 1993. 32, 35

John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the thirteenth national conference on Artificial Intelligence*,

1996. 2, 9, 32, 35, 98

Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*, 2005. 2, 4, 7, 9, 11, 14, 23, 25, 34, 35, 42, 59, 98

Luke S. Zettlemoyer and Michael Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007. 2, 7, 9, 11, 23, 26, 34, 35, 42

Yue Zhang and Stephen Clark. Shift-reduce CCG parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, 2011. 45

GuoDong Zhou, Jian Su, Jie Zhang, and Min Zhang. Exploring various knowledge in relation extraction. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 2005. 47